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

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).

> **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).*

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

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.*

Contents

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

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.

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

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

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.

- 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.

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.

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 $\mathrm{Prob}(X=00)=\frac{1}{3}$ and $\mathrm{Prob}(X=111)=\frac{2}{3}$. 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 $\mathrm{Prob}(f(X)=1)$, 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

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

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

where $\mathrm{val}(E(x,x))$ equals 1 if $E(x,x)$ holds and equals 0 otherwise. For example, for every random variable X, we have $\mathrm{Prob}(X=X)=1$. 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 $\mathrm{Prob}(E(X,Y))$ denotes the probability that $E(x,y)$ holds when the pair (x,y) is chosen with probability $\mathrm{Prob}(X=x) \cdot \mathrm{Prob}(Y=y)$. Namely,

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

For example, for every two independent random variables, X and Y, we have $\mathrm{Prob}(X=Y)=1$ 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, $\mathrm{Prob}(U_n=\alpha)$ 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

$$\mathrm{Prob}\left(X \geq v\right) < \frac{\mathrm{Exp}(X)}{v}$$

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

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

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

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

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

Proof: Define the random variables $\overline{X}_i \stackrel{\text{def}}{=} X_i - \text{Exp}(X_i)$. 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

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

Now (again using the linearity of Exp)

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

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

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

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

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,

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

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.

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

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, $\mathrm{Prob}(M(x)=y)$ is merely the fraction of $r \in \{0,1\}^{t_M(x)}$ for which $M_r(x) = y$. Namely,

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

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*

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

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

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 $\mu'(n) \stackrel{\mathrm{def}}{=} p(n) \cdot \mu(n)$ 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.

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*

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 $\mathcal{P} \subseteq \mathcal{P}/\mathrm{poly}$ (in fact strict containment can be proven by considering non-recursive unary languages). Furthermore,

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

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

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

Assume, without loss of generality, that on each input of length n, machine M uses the same number, $m = \mathrm{poly}(n)$, 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 $x \in \{0,1\}^*$,

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

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 $r_1, r_2, ...$ constructed as above, demonstrates that L is in $\mathcal{P}/\mathrm{poly}$. ■

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

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.

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)

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

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.

Chapter 2

Computational Difficulty

In this chapter we present several variants of the definition of one-way functions. In particular, we define strong and weak one-way functions. We prove that the existence of weak one-way functions imply the existence of strong ones. The proof provides a simple example of a case where a computational statement is much harder to prove than its "information theoretic analogue". Next, we define hard-core predicates, and prove that every one-way function "has" a hard-core predicate.

2.1 One-Way Functions: Motivation

As stated in the introduction chapter, modern cryptography is based on a gap between efficient algorithms guaranteed for the legitimate user versus the computational infeasibility of retrieving protected information for an adversary. To illustrate this, we concentrate on the cryptographic task of secure data communication, namely encryption schemes.

In secure encryption schemes, the legitimate user should be able to easily decipher the messages using some private information available to him, yet an adversary (not having this private information) should not be able to decrypt the ciphertext efficiently (i.e., in probabilistic polynomial-time). On the other hand, a non-deterministic machine can quickly decrypt the ciphertext (e.g., by guessing the private information). Hence, the existence of secure encryption schemes implies that there are tasks (e.g., "breaking" encryption schemes) that can be performed by non-deterministic polynomial-time machines, yet cannot be performed by deterministic (or even randomized) polynomial-time machines. In other words, a necessary condition for the existence of secure encryption schemes is that $\mathcal{NP}$ is not contained in $\mathcal{BPP}$ (and thus $\mathcal{P} \neq \mathcal{NP}$).

Although $\mathcal{P} \neq \mathcal{NP}$ is a necessary condition it is not a sufficient one. $\mathcal{P} \neq \mathcal{NP}$ implies that the encryption scheme is hard to break in the worst case. It does not rule-out the

possibility that the encryption scheme is easy to break almost always. Indeed, one can construct "encryption schemes" for which the breaking problem is NP-complete, and yet there exist an efficient breaking algorithm that succeeds 99% of the time. Hence, worst-case hardness is a poor measure of security. Security requires hardness on most cases or at least "average-case hardness". A necessary condition for the existence of secure encryption schemes is thus the existence of languages in $\mathcal{NP}$ which are hard on the average. It is not known whether $\mathcal{P} \neq \mathcal{NP}$ implies the existence of languages in $\mathcal{NP}$ which are hard on the average.

The mere existence of problems (in NP) which are hard on the average does not suffice either. In order to be able to use such hard-on-the-average problems we must be able to generate hard instances together with auxiliary information which enable to solve these instances fast. Otherwise, these hard instances will be hard also for the legitimate users, and consequently the legitimate users gain no computational advantage over the adversary. Hence, the existence of secure encryption schemes implies the existence of an efficient way (i.e. probabilistic polynomial-time algorithm) of generating instances with corresponding auxiliary input so that

1. it is easy to solve these instances given the auxiliary input; and
2. it is hard on the average to solve these instances (when not given the auxiliary input).

The above requirement is captured by the definition of one-way functions presented in the next subsection. For further details see Exercise 1.

2.2 One-Way Functions: Definitions

In this section, we present several definitions of one-way functions. The first version, hereafter referred to as strong one-way function (or just one-way function), is the most popular one. We also present weak one-way functions, non-uniformly one-way functions, and plausible candidates for such functions.

2.2.1 Strong One-Way Functions

Loosely speaking, a one-way function is a function which is easy to compute but hard to invert. The first condition is quite clear: saying that a function f is easy to compute means that there exists a polynomial-time algorithm that on input x outputs $f(x)$. The second condition requires more elaboration. Saying that a function f is hard to invert means that every probabilistic polynomial-time algorithm trying, on input y to find an inverse of y under f, may succeed only with negligible (in $|y|$) probability. A sequence $\{s_n\}_{n\in\mathbb{N}}$ is called negligible in n if for every polynomial $p(\cdot)$ and all sufficiently large n's it holds that $s_n < \frac{1}{p(n)}$. Further discussion proceeds the definition.

Definition 2.1 (strong one-way functions): *A function* $f : \{0,1\}^* \mapsto \{0,1\}^*$ *is called* (strongly) one-way *if the following two conditions hold*

1. easy to compute: *There exists a (deterministic) polynomial-time algorithm,* A*, so that on input* x *algorithm* A *outputs* $f(x)$ *(i.e.,* $A(x) = f(x)$*).*
2. hard to invert: *For every probabilistic polynomial-time algorithm,* A'*, every polynomial* $p(\cdot)$*, and all sufficiently large* n*'s*

$$\mathrm{Prob}\left(A'(f(U_n), 1^n) \in f^{-1}f(U_n)\right) < \frac{1}{p(n)}$$

Recall that U_n denotes a random variable uniformly distributed over $\{0,1\}^n$. Hence, the probability in the second condition is taken over all the possible values assigned to U_n and all possible internal coin tosses of A', with uniform probability distribution. In addition to an input in the range of f, the inverting algorithm is also given the desired length of the output (in unary notation). The main reason for this convention is to rule out the possibility that a function is consider one-way merely because the inverting algorithm does not have enough time to print the output. Consider for example the function f_{len} defined by $f_{\mathrm{len}}(x) = y$ where y is the binary representation of the length of x (i.e., $f_{\mathrm{len}}(x) = |x|$). Since $|f_{\mathrm{len}}(x)| = \log_2 |x|$ no algorithm can invert $f_{\mathrm{len}}(x)$ in time polynomial in $|f_{\mathrm{len}}(x)|$, yet there exists an obvious algorithm which inverts $f_{\mathrm{len}}(x)$ in time polynomial in $|x|$. In general, the auxiliary input $1^{|x|}$, provided in conjunction to the input $f(x)$, allows the inverting algorithm to run in time polynomial in the total length of the input and the desired output. Note that in the special case of length preserving functions f (i.e., $|f(x)| = |x|$ for all x's), the auxiliary input is redundant.

Hardness to invert is interpreted as an upper bound on the success probability of efficient inverting algorithms. The probability is measured with respect to both the random choices of the inverting algorithm and the distribution of the (main) input to this algorithm (i.e., $f(x)$). The input distribution to the inverting algorithm is obtained by applying f to a uniformly selected $x \in \{0,1\}^n$. If f induces a permutation on $\{0,1\}^n$ then the input to the inverting algorithm is uniformly distributed over $\{0,1\}^n$. However, in the general case where f is not necessarily a one-to-one function, the input distribution to the inverting algorithm may differ substantially from the uniform one. In any case, it is required that the success probability, defined over the above probability space, is negligible (as a function of the length of x), where negligible means being bounded above by all functions of the form $\frac{1}{\mathrm{poly}(n)}$. To further clarify the condition made on the success probability, we consider the following examples.

Consider, an algorithm A_1 that on input $(y, 1^n)$ randomly selects and outputs a string of length n. In case f is a 1-1 function, we have

$$\mathrm{Prob}\left(A_1(f(U_n), 1^n) \in f^{-1}f(U_n)\right) = \frac{1}{2^n}$$

since for every x the probability that $A_1(f(x))$ equals x is exactly 2^{-n}. Hence, the success probability of A_1 on any 1-1 function A_1 is negligible. On the other hand, for every function f, the success probability of A_1 on input $f(U_n)$ is never zero (specifically it is at least 2^{-n}). In case f is constant over strings of the same length (e.g., $f(x) = 0^{|x|}$), we have

$$\mathrm{Prob}\left(A_1(f(U_n), 1^n) \in f^{-1}f(U_n)\right) = 1$$

since every $x \in \{0,1\}^n$ is a preimage of 0^n under f. It follows that a one-way function cannot be constant on strings of the same length. Another trivial algorithm, denoted A_2, is one that computes a function which is constant on all inputs of the same length (e.g., $A_2(y, 1^n) = 1^n$). For every function f we have

$$\mathrm{Prob}\left(A_2(f(U_n), 1^n) \in f^{-1}f(U_n)\right) \geq \frac{1}{2^n}$$

(with equality in case $f(1^n)$ has a single preimage under f). Hence, the success probability of A_2 on any 1-1 function is negligible. On the other hand, if $\mathrm{Prob}(f(U_n) = f(1^n))$ is non-negligible then so is the success probability of algorithm A_2.

A few words, concerning the notion of negligible probability, are in place. The above definition and discussion considers the success probability of an algorithm to be *negligible* if, as a function of the input length, the success probability is bounded above by every polynomial fraction. It follows that repeating the algorithm polynomially (in the input length) many times yields a new algorithm that also has a negligible success probability. In other words, events which occur with negligible (in n) probability remain negligible even if the experiment is repeated for polynomially (in n) many times. Hence, defining negligible success as "occurring with probability smaller than any polynomial fraction" is naturally coupled with defining feasible as "computed within polynomial time".

A "strong negation" of the notion of a negligible fraction/probability is the notion of a non-negligible fraction/probability. We say that a function ν is *non-negligible* if there exists a polynomial $p(\cdot)$ such that for all sufficiently large n's it holds that $\mu(n) > \frac{1}{p(n)}$. Note that functions may be neither negligible nor non-negligible.

2.2.2 Weak One-Way Functions

One-way functions as defined above, are one-way in a very strong sense. Namely, any efficient inverting algorithm has negligible success in inverting them. A much weaker definition, presented below, only requires that all efficient inverting algorithm fails with some non-negligible probability.

Definition 2.2 (weak one-way functions): *A function $f : \{0,1\}^* \mapsto \{0,1\}^*$ is called* weakly one-way *if the following two conditions hold*

1. easy to compute: *as in the definition of strong one-way function.*

2. slightly-hard to invert: *There exists a polynomial $p(\cdot)$ such that for every probabilistic polynomial-time algorithm, A', and all sufficiently large n's*

$$\text{Prob}\left(A'(f(U_n), 1^n) \notin f^{-1}f(U_n)\right) > \frac{1}{p(n)}$$

2.2.3 Two Useful Length Conventions

In the sequel it will be convenient to use the following two conventions regarding the *length* of the of the preimages and images of a one-way function. In the current subsection we justify the used of these conventions.

One-way functions defined only for some lengths

In many cases it is more convenient to consider one-way functions with domain partial to the set of all strings. In particular, this facilitates the introduction of some structure in the domain of the function. A particularly important case, used throughout the rest of this section, is that of functions with domain $\cup_{n\in\mathbb{N}}\{0,1\}^{p(n)}$, where $p(\cdot)$ is some polynomial. Let $I \subseteq \mathbb{N}$, and denote by $s_I(n)$ the successor of n with respect to I; namely, $s_I(n)$ is the smallest integer that is both greater than n and in the set I (i.e., $s_I(n) \stackrel{\text{def}}{=} \min\{i \in I : i > n\}$). A set $I \subseteq \mathbb{N}$ is called *polynomial-time enumerable* if there exists an algorithm that on input n, halts within $\text{poly}(n)$ steps and outputs $s_I(n)$. Let I be a polynomial-time enumerable set and f be a function with domain $\cup_{n\in I}\{0,1\}^n$. We call f strongly (resp. weakly) *one-way on lengths in I* if f is polynomial-time computable and is hard to invert over n's in I. Such one-way functions can be easily modified into function with the set of all strings as domain, while preserving one-wayness and some other properties of the original function. In particular, for any function f with domain $\cup_{n\in I}\{0,1\}^n$, we can construct a function $g : \{0,1\}^* \mapsto \{0,1\}^*$ by letting

$$g(x) \stackrel{\text{def}}{=} f(x')$$

where x' is the longest prefix of x with length in I. (In case the function f is length preserving, see definition below, we can preserve this property by modifying the construction so that $g(x) \stackrel{\text{def}}{=} f(x')x''$ where $x = x'x''$, and x' is the longest prefix of x with length in I. The following proposition remains valid also in this case, with a minor modification in the proof.)

Proposition 2.3 : *Let I be a polynomial-time enumerable set, and f be strongly (resp. weakly) one-way on lengths in I. Then g (constructed above) is strongly (resp. weakly) one-way (in the ordinary sense).*

Although the validity of the above proposition is very appealing, we urge the reader not to skip the following proof. The proof, which is indeed quite simple, uses for the first time in this book an argument that is used extensively in the sequel. The argument used to prove the "hardness to invert" property of the function g proceeds by assuming, to the contradiction, that g can be efficiently inverted with unallowable success probability. Contradiction is derived by deducing that f can be efficiently inverted with unallowable success probability. In other words, inverting f is "reduced" to inverting g. The term "reduction" is used here in a non-standard sense, which preserves the success probability of the algorithms. This kind of an argument is called a *reducibility argument*.

Proof: We first prove that g can be computed in polynomial-time. To this end we use the fact that I is a polynomial-time enumerable set. It follows that on input x one can find in polynomial-time the largest $m \leq |x|$ that satisfies $m \in I$. Computing $g(x)$ amounts to finding this m, and applying the function f to the m-bit prefix of x.

We next prove that g maintains the "hardness to invert" property of f. For sake of concreteness we present here only the proof for the case that f is strongly one-way. The proof for the case that f is weakly one-way is analogous.

The prove proceeds by contradiction. We assume, on contrary to the claim (of the proposition), that there exists an efficient algorithm that inverts g with success probability that is not negligible. We use this inverting algorithm (for g) to construct an efficient algorithm that inverts f with success probability that is not negligible, hence deriving a contradiction (to the hypothesis of the proposition). In other words, we show that inverting f (with unallowable success probability) is efficiently reducible to inverting g (with unallowable success probability), and hence conclude that the latter is not feasible. The reduction is based on the observation that inverting g on images of arbitrary length yields inverting g also on images of length in I, and that on such lengths g collides with f. Details follow.

Given an algorithm, B', for inverting g we construct an algorithm, A', for inverting f so that A' has complexity and success probability related to that of B'. Algorithm A' uses algorithm B' as a subroutine and proceeds as follows. On input y and 1^n (supposedly y is in the range of $f(U_n)$ and $n \in I$) algorithm A' first computes $\mathrm{s}_I(n)$ and sets $k \stackrel{\mathrm{def}}{=} \mathrm{s}_I(n) - n - 1$. For every $0 \leq i \leq k$, algorithm A' initiates algorithm B', on input $(y, 1^{n+i})$, obtaining $z_i \leftarrow B'(y, 1^{n+i})$, and checks if $g(z_i) = y$. In case one of the z_i's satisfies the above condition, algorithm A' outputs the n-bit long prefix of z_i. This prefix is in the preimage of y under f (since $g(x'x'') = f(x')$ for all $x' \in \{0,1\}^n$ and $|x''| \leq k$). Clearly, if B' is a probabilistic polynomial-time algorithm then so is A'. We now analyze the success probability of A' (showing that if B' inverts g with unallowable success probability then A' inverts f with unallowable success probability).

Suppose now, on the contrary to our claim, that g is not strongly one-way, and let B' be an algorithm demonstrating this contradiction hypothesis. Namely, there exists a polynomial $p(\cdot)$ so that for infinitely many m's the probability that B' inverts g on $g(U_m)$

is at least $\frac{1}{p(m)}$. Let us denote the set of these m's by M. Define a function $\ell_I : \mathbb{N} \mapsto I$ so that $\ell_I(m)$ is the largest lower bound of m in I (i.e., $\ell_I(m) \stackrel{\text{def}}{=} \max\{i \in I : i \leq m\}$). Clearly, $m \leq \mathrm{s}_I(\ell_I(m)) - 1$ for every m. The following two claims relate the success probability of algorithm A' with that of algorithm B'.

Claim 2.3.1: Let m be an integer and $n = \ell_I(m)$. Then

$$\mathrm{Prob}\left(A'(f(U_n), 1^n) \in f^{-1}f(U_n)\right) \geq \mathrm{Prob}\left(B'(g(U_m), 1^m) \in g^{-1}g(U_m)\right)$$

(Namely, the success probability of algorithm A' on $f(U_{\ell_I(m)})$ is bounded below by the success probability of algorithm B' on $g(U_m)$.)

Proof: By construction of A', on input $(f(x'), 1^n)$, where $x' \in \{0,1\}^n$, algorithm A' obtains the value $B'(f(x'), 1^t)$, for every $t \leq \mathrm{s}_I(n) - 1$. In particular, since $m \leq \mathrm{s}_I(\ell_I(m)) - 1 = \mathrm{s}_I(n) - 1$, it follows that algorithm A' obtains the value $B'(f(x'), 1^m)$. By definition of g, for all $x'' \in \{0,1\}^{m-n}$, it holds that $f(x') = g(x'x'')$. The claim follows. □

Claim 2.3.2: There exists a polynomial $q(\cdot)$ such that $m < q(\ell_I(m))$, for all m's.
Hence, the set $S \stackrel{\text{def}}{=} \{\ell_I(m) : m \in M\}$ is infinite.

Proof: Using the polynomial-time enumerability of I, we get $\mathrm{s}_I(n) < \mathrm{poly}(n)$, for every n. Therefore, for every m, we have $m < \mathrm{s}_I(\ell_I(m)) < \mathrm{poly}(\ell_I(m))$. Furthermore, S must be infinite, otherwise for n upper-bounding S we get $m < q(n)$ for every $m \in M$.□

Using Claims 2.3.1 and 2.3.2, it follows that, for every $n = \ell_I(m) \in S$, the probability that A' inverts f on $f(U_n)$ is at least $\frac{1}{p(m)} > \frac{1}{p(q(n))} = \frac{1}{\mathrm{poly}(n)}$. It follows that f is not strongly one-way, in contradiction to the proposition's hypothesis. ■

Length-regular and length-preserving one-way functions

A second useful convention is to assume that the function, f, we consider is *length regular* in the sense that, for every $x, y \in \{0,1\}^*$, if $|x| = |y|$ then $|f(x)| = |f(y)|$. We point out that the transformation presented above preserves length regularity. A special case of length regularity, preserved by a the modified transformation presented above, is of *length preserving* functions.

Definition 2.4 (length preserving functions): *A function f is* length preserving *if for every $x \in \{0,1\}^*$ it holds that $|f(x)| = |x|$.*

Given a strongly (resp. weakly) one-way function f, we can construct a strongly (resp. weakly) one-way function h which is length preserving, as follows. Let p be a polynomial bounding the length expansion of f (i.e., $|f(x)| \leq p(|x|)$). Such a polynomial must exist

since f is polynomial-time computable. We first construct a length regular function g by defining

$$g(x) \stackrel{\text{def}}{=} f(x)10^{p(|x|)-|f(x)|}$$

(We use a padding of the form 10^* in order to facilitate the parsing of $g(x)$ into $f(x)$ and the "leftover" padding.) Next, we define h only on strings of length $p(n)+1$, for $n \in \mathbb{N}$, by letting

$$h(x'x'') \stackrel{\text{def}}{=} g(x') \text{ , where } |x'x''| = p(|x'|) + 1$$

Clearly, h is length preserving.

Proposition 2.5 : *If f is a strongly (resp. weakly) one-way function then so are g and h (constructed above).*

Proof Sketch: It is quite easy to see that both g and h are polynomial-time computable. Using "reducibility arguments" analogous to the one used in the previous proof, we can establish the hardness-to-invert of both g and h. For example, given an algorithm B' for inverting g, we construct an algorithm A' for inverting f as follows. On input y and 1^n (supposedly y is in the range of $f(U_n)$), algorithm A' halts with output $B'(y10^{p(n)-|y|}, 1^{p(n)+1})$. ■

The reader can easily verify that if f is length preserving then it is redundant to provide the inverting algorithm with the auxiliary input $1^{|x|}$ (in addition to $f(x)$). The same holds if f is length regular and does not shrink its input by more than a polynomial factor (i.e., there exists a polynomial $p(\cdot)$ such that $p(|f(x)|) \geq |x|$ for all x). In the sequel, we will only deal with one-way functions that are length regular and does not shrink their its input by more that a polynomial factor. Furthermore, we will mostly deal with length preserving functions. Hence, in these cases, *we assume, without loss of generality, that the inverting algorithm is only given $f(x)$ as input.*

Functions which are length preserving are not necessarily 1-1. Furthermore, the assumption that 1-1 one-way functions exist seems stronger than the assumption that arbitrary (and hence length preserving) one-way functions exist. For further discussion see Section 2.4.

2.2.4 Candidates for One-Way Functions

Following are several candidates for one-way functions. Clearly, it is not known whether these functions are indeed one-way. This is only a conjecture supported by extensive research which has so far failed to produce an efficient inverting algorithm (having non-negligible success probability).

Integer factorization

In spite of the extensive research directed towards the construction of efficient (integer) factoring algorithms, the best algorithms known for factoring an integer N, run in time $L(P) \stackrel{\text{def}}{=} 2^{O(\sqrt{\log P \log\log P})}$, where P is the second biggest prime factor of N. Hence it is reasonable to believe that the function f_{mult}, which partitions its input string into two parts and returns the (binary representation of the) integer resulting by multiplying (the integers represented by) these parts, is one-way. Namely, let

$$f_{\text{mult}}(x,y) = x \cdot y$$

where $|x| = |y|$ and $x \cdot y$ denotes (the string representing) the integer resulting by multiplying the integers (represented by the strings) x and y. Clearly, f_{mult} can be computed in polynomial-time. Assuming the intractability of factoring and using the "density of primes" theorem (which guarantees that at least $\frac{N}{\log_2 N}$ of the integers smaller than N are primes) it follows that f_{mult} is at least weakly one-way. Using a more sophisticated argument, one can show that f_{mult} is strongly one-way. Other popular functions (e.g. the RSA) related to integer factorization are discussed in Subsection 2.4.3.

Decoding of random linear codes

One of the most outstanding open problems in the area of error correcting codes is that of presenting efficient decoding algorithms for random linear codes. Of particular interest are random linear codes with constant information rate which can correct a constant fraction of errors. An (n,k,d)-*linear-code* is a k-by-n binary matrix in which the vector sum (mod 2) of any non-empty subset of rows results in a vector with at least d one-entries. (A k-bit long message is encoded by multiplying it with the k-by-n matrix, and the resulting n-bit long vector has a unique preimage even when flipping up to $\frac{d}{2}$ of its entries.) The Gilbert-Varshanov Bound for linear codes guarantees the existence of such a code, provided that $\frac{k}{n} < 1 - H_2(\frac{d}{n})$, where $H_2(p) \stackrel{\text{def}}{=} -p\log_2 p - (1-p)\log_2(1-p)$ if $p < \frac{1}{2}$ and $H_2(p) \stackrel{\text{def}}{=} 1$ otherwise (i.e., $H_2(\cdot)$ is a modification of the binary entropy function). Similarly, if for some $\epsilon > 0$ it holds that $\frac{k}{n} < 1 - H_2(\frac{(1+\epsilon)d}{n})$ then almost all k-by-n binary matrices constitute (n,k,d)-linear-codes. Consider three constants $\kappa, \delta, \epsilon > 0$ satisfying $\kappa < 1 - H_2((1+\epsilon)\delta)$. The function f_{code}, hereafter defined, seems a plausible candidate for a one-way function.

$$f_{\text{code}}(C,x,i) \stackrel{\text{def}}{=} (C, xC + e(i))$$

where C is an κn-by-n binary matrix, x is a κn-dimensional binary vector, i is the index of an n-dimensional binary vector having at most $\frac{\delta n}{2}$ one-entries (the string itself is denoted $e(i)$), and the arithmetic is in the n-dimensional binary vector space. Clearly, f_{code} is polynomial-time computable. An efficient algorithm for inverting f_{code} would yield an efficient algorithm for inverting a non-negligible fraction of the linear codes (an earthshaking result in coding theory).

The subset sum problem

Consider the function $f_{\rm ss}$ defines as follows.

$$f_{\rm ss}(x_1, ..., x_n, I) = (x_1, ..., x_n, \sum_{i \in I} x_i)$$

where $|x_1| = \cdots = |x_n| = n$, and $I \subseteq \{1, 2, ..., n\}$. Clearly, $f_{\rm ss}$ is polynomial-time computable. The fact that the subset-sum problem is NP-complete cannot serve as evidence to the one-wayness of $f_{\rm ss}$. On the other hand, the fact that the subset-sum problem is easy for special cases (such as having "hidden structure" and/or "low density") can not serve as evidence for the weakness of this proposal. The conjecture that $f_{\rm ss}$ is one-way is based on the failure of known algorithm to handle random "high density" instances (i.e., instances in which the length of the elements is not greater than their number). Yet, one has to admit that the evidence in favour of this candidate is much weaker than the evidence in favour of the two previous ones.

2.2.5 Non-Uniformly One-Way Functions

In the above two definitions of one-way functions the inverting algorithm is probabilistic polynomial-time. Stronger versions of both definitions require that the functions cannot be inverted even by non-uniform families of polynomial-size circuits. We stress that the "easy to compute" condition is still stated in terms of uniform algorithms. For example, following is a non-uniform version of the definition of strong one-way functions.

Definition 2.6 (non-uniformly strong one-way functions): *A function $f : \{0,1\}^* \mapsto \{0,1\}^*$ is called* non-uniformly one-way *if the following two conditions hold*

1. easy to compute: *There exists a (deterministic) polynomial-time algorithm, A, so that on input x algorithm A outputs $f(x)$ (i.e., $A(x) = f(x)$).*
2. hard to invert: *For every (even non-uniform) family of polynomial-size circuits, $\{C_n\}_{n \in \mathbb{N}}$, every polynomial $p(\cdot)$, and all sufficiently large n's*

$$\text{Prob}\left(C_n(f(U_n)) \in f^{-1}f(U_n)\right) < \frac{1}{p(n)}$$

The probability in the second condition is taken only over all the possible values of U_n. Note that it is redundant to give 1^n as an auxiliary input to C_n.

It can be shown that if f is non-uniformly one-way then it is one-way (i.e., in the uniform sense). The proof follows by converting any (uniform) probabilistic polynomial-time

inverting algorithm into a non-uniform family of polynomial-size circuits, without decreasing the success probability. Details follow. Let A' be a probabilistic polynomial-time (inverting) algorithm. Let r_n denote a sequence of coin tosses for A' maximizing the success probability of A'. Namely, r_n satisfies $\mathrm{Prob}(A'_{r_n}(f(U_n)) \in f^{-1}f(U_n)) \geq \mathrm{Prob}(A(f(U_n)) \in f^{-1}f(U_n))$, where the first probability is taken only over all possible values of U_n and the second probability is also over all possible coin tosses for A'. (Recall that $A'_r(y)$ denotes the output of algorithm A' on input y and internal coin tosses r.) The desired circuit C_n incorporates the code of algorithm A' and the sequence r_n (which is of length polynomial in n).

It is possible that one-way functions exist (in the uniform sense) and yet there are no non-uniformly one-way functions. However, such a possibility is considered not very plausible.

2.3 Weak One-Way Functions Imply Strong Ones

We first remark that not every weak one-way function is necessarily a strong one. Consider for example a one-way function f (which without loss of generality is length preserving). Modify f into a function g so that $g(x,p) = (f(x),p)$ if p starts with $\log_2 |x|$ zeros and $g(x,p) = (x,p)$ otherwise, where (in both cases) $|x| = |p|$. We claim that g is a weak one-way function but not a strong one. Clearly, g can not be a strong one-way function (since for all but a $\frac{1}{n}$ fraction of the strings of length $2n$ the function g coincides with the identity function). To prove that g is weakly one-way we use a "reducibility argument". Details follow.

Proposition 2.7 *Let f be a one-way function (even in the weak sense). Then g, constructed above, is a weakly one-way function.*

Proof: Given a probabilistic polynomial-time algorithm, B', for inverting g, we construct a probabilistic polynomial-time algorithm A' which inverts f with "related" success probability. Following is the description of algorithm A'. On input y, algorithm A' sets $n \stackrel{\mathrm{def}}{=} |y|$ and $l \stackrel{\mathrm{def}}{=} \log_2 n$, selects p' uniformly in $\{0,1\}^{n-l}$, computes $z \stackrel{\mathrm{def}}{=} B'(y, 0^l p')$, and halts with output the n-bit prefix of z. Let S_{2n} denote the sets of all $2n$-bit long strings which start with $\log_2 n$ zeros (i.e., $s_{2n} \stackrel{\mathrm{def}}{=} \{0^{\log_2 n}\alpha : \alpha \in \{0,1\}^{2n-\log_2 n}\}$). Then, by construction of A' and g, we have

$$\begin{aligned}
\mathrm{Prob}\left(A'(f(U_n)) \in f^{-1}f(U_n)\right) &\geq \mathrm{Prob}\left(B'(f(U_n), 0^l U_{n-l}) \in (f^{-1}f(U_n), 0^l U_{n-l})\right) \\
&= \mathrm{Prob}\left(B'(g(U_{2n})) \in g^{-1}g(U_{2n}) \mid U_{2n} \in S_{2n}\right) \\
&\geq \frac{\mathrm{Prob}\left(B'(g(U_{2n})) \in g^{-1}g(U_{2n})\right) - \mathrm{Prob}\left(U_{2n} \notin S_{2n}\right)}{\mathrm{Prob}\left(U_{2n} \in S_{2n}\right)}
\end{aligned}$$

$$\begin{aligned} &= \frac{1}{n} \cdot \left(\text{Prob}\left(B'(g(U_{2n})) \in g^{-1}g(U_{2n}) \right) - \left(1 - \frac{1}{n} \right) \right) \\ &= 1 - n \cdot \left(1 - \text{Prob}\left(B'(g(U_{2n})) \in g^{-1}g(U_{2n}) \right) \right) \end{aligned}$$

(For the second inequality, we used $\text{Prob}(A|B) = \frac{\text{Prob}(A \cap B)}{\text{Prob}(B)}$ and $\text{Prob}(A \cap B) \geq \text{Prob}(A) - \text{Prob}(\overline{B})$.) It should not come as a surprise that the above expression is meaningful only in case $\text{Prob}(B'(g(U_{2n})) \in g^{-1}g(U_{2n})) > -(1 - \frac{1}{n})$.

It follows that, for every polynomial $p(\cdot)$ and every integer n, if B' inverts g on $g(U_{2n})$ with probability greater than $1 - \frac{1}{p(2n)}$ then A' inverts f on $f(U_n)$ with probability greater than $1 - \frac{n}{p(2n)}$. Hence, if g is not weakly one-way (i.e., for every polynomial $p(\cdot)$ there exist infinitely many m's such that g can be inverted on $g(U_m)$ with probability $\geq 1 - 1/p(m)$) then also f is not weakly one-way (i.e., for every polynomial $q(\cdot)$ there exist infinitely many n's such that f can be inverted on $f(U_n)$ with probability $\geq 1 - 1/q(n)$). This contradicts our hypothesis (that f is one-way). ■

We have just shown that, unless no one-way functions exist, there exist weak one-way functions which are not strong ones. Fortunately, we can rule out the possibility that all one-way functions are only weak ones. In particular, the existence of weak one-way functions implies the existence of strong ones.

Theorem 2.8 : *Weak one-way functions exist if and only if strong one-way functions exist.*

We strongly recommend to the reader not to skip the following proof, since we believe that the proof is very instructive to the rest of the book. In particular, the proof demonstrates that amplification of computational difficulty is much more involved than amplification of an analogous probabilistic event.

Proof: Let f be a weak one-way function, and let p be the polynomial guaranteed by the definition of a weak one-way function. Namely, every probabilistic polynomial-time algorithm fails to invert f on $f(U_n)$ with probability at least $\frac{1}{p(n)}$. We assume, for simplicity, that f is length preserving (i.e. $|f(x)| = |x|$ for all x's). This assumption, which is not really essential, is justified by Proposition 2.5. We define a function g as follows

$$g(x_1, ..., x_{t(n)}) \stackrel{\text{def}}{=} f(x_1), ..., f(x_{t(n)})$$

where $|x_1| = |x_{t(n)}| = n$ and $t(n) \stackrel{\text{def}}{=} n \cdot p(n)$. Namely, the $n^2p(n)$-bit long input of g is partitioned into $t(n)$ blocks each of length n, and f is applied to each block.

Clearly, g can be computed in polynomial-time (by an algorithm which breaks the input into blocks and applies f to each block). Furthermore, it is easy to see that inverting g on

$g(x_1, ..., x_{t(n)})$ requires finding a preimage to each $f(x_i)$. One may be tempted to deduce that it is also clear that g is a strongly one-way function. An naive argument, assumes implicitly (with no justification) that the inverting algorithm works separately on each $f(x_i)$. If this were indeed the case then the probability that an inverting algorithm successfully inverts all $f(x_i)$'s is at most $(1 - \frac{1}{p(n)})^{n \cdot p(n)} < 2^{-n}$ (which is negligible also as a function of $n^2 p(n)$). However, the assumption that an algorithm trying to invert g works independently on each $f(x_i)$ cannot be justified. Hence, a more complex argument is required.

Following is an outline of our proof. The proof that g is strongly one-way proceeds by a contradiction argument. We assume on the contrary that g is not strongly one-way; namely, we assume that there exists a polynomial-time algorithm that inverts g with probability which is not negligible. We derive a contradiction by presenting a polynomial-time algorithm which, for infinitely many n's, inverts f on $f(U_n)$ with probability greater than $1 - \frac{1}{p(n)}$ (in contradiction to our hypothesis). The inverting algorithm for f uses the inverting algorithm for g as a subroutine (without assuming anything about the manner in which the latter algorithm operates). Details follow.

Suppose that g is not strongly one-way. By definition, it follows that there exists a probabilistic polynomial-time algorithm B' and a polynomial $q(\cdot)$ so that for infinitely many m's

$$\text{Prob}\left(B'(g(U_m)) \in g^{-1}g(U_m)\right) > \frac{1}{q(m)}$$

Let us denote by M', the infinite set of integers for which the above holds. Let N' denote the infinite set of n's for which $n^2 \cdot p(n) \in M'$ (note that all m's considered are of the form $n^2 \cdot p(n)$, for some integer n).

We now present a probabilistic polynomial-time algorithm, A', for inverting f. On input y (supposedly in the range f) algorithm A' proceeds by applying the following probabilistic procedure, denoted I, on input y for $a(|y|)$ times, where $a(\cdot)$ is a polynomial depends on the polynomials p and q (specifically, we set $a(n) \stackrel{\text{def}}{=} 2n^2 \cdot p(n) \cdot q(n^2 p(n))$).

Procedure I
Input: y (denote $n \stackrel{\text{def}}{=} |y|$).
`For` $i = 1$ `to` $t(n)$ `do begin`

1. Select uniformly and independently a sequence of strings $x_1, ..., x_{t(n)} \in \{0,1\}^n$.
2. Compute

   $$(z_1, ..., z_{t(n)}) \leftarrow B'(f(x_1), ..., f(x_{i-1}), y, f(x_{i+1}), ..., f(x_{t(n)}))$$

   (Note that y is placed in the i^{th} position instead of $f(x_i)$.)
3. If $f(z_i) = y$ then halt and output y.
   (This is considered a *success*).

`end`

We now present a lower bound on the success probability of algorithm A'. To this end we define a set S_n, which contains all n-bit strings on which the procedure I succeeds with non-negligible probability (specifically greater than $\frac{n}{a(n)}$). (The probability is taken only over the coin tosses of algorithm A'). Namely,

$$S_n \stackrel{\text{def}}{=} \left\{ x : \text{Prob}\left(I(f(x)) \in f^{-1}f(x) \right) > \frac{n}{a(n)} \right\}$$

In the next two claims we shall show that S_n contains all but a $\frac{1}{2p(n)}$ fraction of the strings of length $n \in N'$, and that for each string $x \in S_n$ the algorithm A' inverts f on $f(x)$ with probability exponentially close to 1. It will follow that A' inverts f on $f(U_n)$, for $n \in N'$, with probability greater than $1 - \frac{1}{p(n)}$, in contradiction to our hypothesis.

Claim 2.8.1: For every $x \in S_n$

$$\text{Prob}\left(A'(f(x) \in f^{-1}f(x) \right) > 1 - \frac{1}{2^n}$$

Proof: By definition of the set S_n, the procedure I inverts $f(x)$ with probability at least $\frac{n}{a(n)}$. Algorithm A' merely repeats I for $a(n)$ times, and hence

$$\text{Prob}\left(A'(f(x) \notin f^{-1}f(x) \right) < \left(1 - \frac{n}{a(n)} \right)^{a(n)} < \frac{1}{2^n}$$

The claim follows. □

Claim 2.8.2: For every $n \in N'$,

$$|S_n| > \left(1 - \frac{1}{2p(n)} \right) \cdot 2^n$$

Proof: We assume, to the contrary, that $|S_n| \leq (1 - \frac{1}{2p(n)}) \cdot 2^n$. We shall reach a contradiction to our hypothesis concerning the success probability of B'. Recall that by this hypothesis

$$s(n) \stackrel{\text{def}}{=} \text{Prob}\left(B'(g(U_{n^2p(n)})) \in g^{-1}g(U_{n^2p(n)}) \right) > \frac{1}{q(n^2p(n))}$$

Let $U_n^{(1)}, ..., U_n^{(n \cdot p(n))}$ denote the n-bit long blocks in the random variable $U_{n^2p(n)}$ (i.e., these $U_n^{(i)}$'s are independent random variables each uniformly distributed in $\{0,1\}^n$). Clearly, $s(n)$ is the sum of $s_1(n)$ and $s_2(n)$ defined by

$$s_1(n) \stackrel{\text{def}}{=} \text{Prob}\left(B'(g(U_{n^2p(n)})) \in g^{-1}g(U_{n^2p(n)}) \wedge \left(\exists i \text{ s.t. } U_n^{(i)} \notin S_n \right) \right)$$

and

$$s_2(n) \stackrel{\rm def}{=} \mathrm{Prob}\left(B'(g(U_{n^2p(n)})) \in g^{-1}g(U_{n^2p(n)}) \wedge \left(\forall i : U_n^{(i)} \in S_n\right)\right)$$

(Use $\mathrm{Prob}(A) = \mathrm{Prob}(A \wedge B) + \mathrm{Prob}(A \wedge \neg B)$.) We derive a contradiction to the lower bound on $s(n)$ by presenting upper bounds for both $s_1(n)$ and $s_2(n)$ (which sum up to less).

First, we present an upper bound on $s_1(n)$. By the construction of algorithm I it follows that, for every $x \in \{0,1\}^n$ and every $1 \leq i \leq n \cdot p(n)$, the probability that I inverts f on $f(x)$ in the $i^{\rm th}$ iteration equals the probability that B' inverts g on $g(U_{n^2p(n)})$ when $U_n^{(i)} = x$. It follows that, for every $x \in \{0,1\}^n$ and every $1 \leq i \leq n \cdot p(n)$,

$$\mathrm{Prob}\left(I(f(x)) \in f^{-1}f(x)\right) \geq \mathrm{Prob}\left(B'(g(U_{n^2p(n)})) \in g^{-1}g(U_{n^2p(n)}) \mid U_n^{(i)} = x\right)$$

Using trivial probabilistic inequalities (such as $\mathrm{Prob}(\exists i\ A_i) \leq \sum_i \mathrm{Prob}(A_i)$ and $\mathrm{Prob}(A \wedge B) \leq \mathrm{Prob}(A \mid B)$), it follows that

$$\begin{aligned}
s_1(n) &\leq \sum_{i=1}^{n\cdot p(n)} \mathrm{Prob}\left(B'(g(U_{n^2p(n)})) \in g^{-1}g(U_{n^2p(n)}) \wedge U_n^{(i)} \notin S_n\right) \\
&\leq \sum_{i=1}^{n\cdot p(n)} \mathrm{Prob}\left(B'(g(U_{n^2p(n)})) \in g^{-1}g(U_{n^2p(n)}) \mid U_n^{(i)} \notin S_n\right) \\
&\leq \sum_{i=1}^{n\cdot p(n)} \mathrm{Prob}\left(I(f(U_n)) \in f^{-1}f(U_n) \mid U_n \notin S_n\right) \\
&\leq n \cdot p(n) \cdot \frac{n}{a(n)}
\end{aligned}$$

(The last inequality uses the definition of S_n.)

We now present an upper bound on $s_2(n)$. Recall that by the contradiction hypothesis, $|S_n| \leq (1 - \frac{1}{2p(n)}) \cdot 2^n$. It follows that

$$\begin{aligned}
s_2(n) &\leq \mathrm{Prob}\left(\forall i : U_n^{(i)} \in S_n\right) \\
&\leq \left(1 - \frac{1}{2p(n)}\right)^{n\cdot p(n)} \\
&< \frac{1}{2^{\frac{n}{2}}}
\end{aligned}$$

Hence, on one hand $s_1(n) + s_2(n) < \frac{2n^2 \cdot p(n)}{a(n)} = \frac{1}{q(n^2p(n))}$ (equality by definition of $a(n)$). Yet, on the other hand $s_1(n) + s_2(n) = s(n) > \frac{1}{q(n^2p(n))}$. Contradiction is reached and the claim follows. □

Combining Claims 2.8.1 and 2.8.2, It follows that the probabilistic polynomial-time algorithm, A', inverts f on $f(U_n)$, for $n \in N'$, with probability greater than $1 - \frac{1}{p(n)}$, in contradiction to our hypothesis (that f cannot be efficiently inverted with such success probability). The theorem follows. ■

Let us summarize the structure of the proof of Theorem 2.8. Given a weak one-way function f, we first constructed a polynomial-time computable function g. This was done with the intention of later proving that g is strongly one-way. To prove that g is strongly one-way we used a "reducibility argument". The argument transforms efficient algorithms which supposedly contradict the strong one-wayness of g into efficient algorithms which contradict the hypothesis that f is weakly one-way. Hence g must be strongly one-way. We stress that our algorithmic transformation, which is in fact a randomized Cook reduction, makes no implicit or explicit assumptions about the structure of the prospective algorithms for inverting g. Such assumptions, as the "natural" assumption that the inverter of g works independently on each block, cannot be justified (at least not at the current state of understanding of the nature of efficient computations).

Theorem 2.8 has a natural information theoretic (or "probabilistic") analogue which asserts that repeating an experiment, which has a non-negligible success probability, sufficiently many times yields success with very high probability. The reader is probably convinced at this stage that the proof of Theorem 2.8 is much more complex that the proof of the information theoretic analogue. In the information theoretic context the repeated events are independent by definition, whereas in our computational context no such independence can be guaranteed. Another indication to the difference between the two settings follows. In the information theoretic setting the probability that none of the events occur decreases exponentially in the number of repetitions. However, in the computational setting we can only reach a negligible bounds on the inverting probabilities of polynomial-time algorithms. Furthermore, it may be the case that g constructed in the proof of Theorem 2.8 can be efficiently inverted on $g(U_{n^{2p(n)}})$ with success probability which is subexponentially decreasing (e.g., with probability $2^{-\log_2^3 m}$), whereas the analogous information theoretic experiment fails with probability at most 2^{-n}.

By Theorem 2.8, whenever assuming the existence of one-way functions, there is no need to specify whether we refer to weak or strong ones. Thus, as far as the mere existence of one-way function goes, the notions of weak and strong one-way functions are equivalent. However, as far as efficiency considerations are concerned the two notions are not really equivalent, since the above transformation of weak one-way functions into strong ones is not practical. An alternative transformation which is much more efficient does exist for the case of one-way permutations and other specific classes of one-way functions. Further details are presented in Section 2.6.

2.4 One-Way Functions: Variations

In this section, we discuss several issues concerning one-way functions. In the first subsection, we present a function that is (strongly) one-way, provided that one-way functions exist. The construction of this function is of strict abstract interest. In contrast, the issues discussed in the other subsections are of practical importance. First, we present a formulation which is better suited for describing many natural candidates for one-way functions, and use it in order to describe popular candidates for one-way functions. Next, we use this formulation to present one-way functions with additional properties; specifically, (one-way) trapdoor permutations, and clawfree functions. We remark that these additional properties are used in several constructions (e.g., trapdoor permutations are used in the construction of public-key encryption schemes whereas clawfree permutations are used in the construction of collision-free hashing). We conclude this section with remarks addressing the "art" of proposing candidates for one-way functions.

2.4.1 * Universal One-Way Function

Using the result of the previous section and the notion of a universal machine it is possible to prove the existence of a universal one-way function.

Proposition 2.9 *There exists a polynomial-time computable function which is (strongly) one-way if and only if one-way functions exist.*

Proof: A key observation is that there exist one-way functions if and only if there exist one-way functions which can be evaluated by a quadratic time algorithm. (The choice of the specific time bound is immaterial, what is important is that such a specific time bound exists.) This statement is proven using a padding argument. Details follow.

Let f be an arbitrary one-way function, and let $p(\cdot)$ be a polynomial bounding the time complexity of an algorithm for computing f. Define $g(x'x'') \stackrel{\text{def}}{=} f(x')x''$, where $|x'x''| = p(|x'|)$. An algorithm computing g first parses the input into x' and x'' so that $|x'x''| = p(|x'|)$, and then applies f on x'. The parsing and the other overhead operations can be implemented in quadratic time (in $|x'x''|$), whereas computing $f(x')$ is done within time $p(|x'|) = |x'x''|$ (which is linear in the input length). Hence, g can be computed (by a Turing machine) in quadratic time. The reader can verify that g is one-way using a "reducibility argument" analogous to the one used in the proof of Proposition 2.5.

We now present a (universal one-way) function, denoted f_{uni}.

$$f_{\text{uni}}(\text{desc}(M), x) \stackrel{\text{def}}{=} (\text{desc}(M), M(x))$$

where $\mathrm{desc}(M)$ is a description of Turing machine M, and $M(x)$ is defined as the output of M on input x if M runs at most quadratic time on x, and as x otherwise. Clearly, f_{uni} can be computed in polynomial-time by a universal machine which uses a step counter. To show that f_{uni} is one-way we use a "reducibility argument". By the above observation, we know that there exist a one-way function g which is computed in quadratic time. Let M_g be the quadratic time machine computing g. Clearly, an (efficient) algorithm inverting f_{uni} on inputs of the form $f_{\mathrm{uni}}(\mathrm{desc}(M_g), U_n)$, with probability $\epsilon(n)$, can be easily modified into an (efficient) algorithm inverting g on inputs of the form $g(U_n)$, with probability $\epsilon(n)$. It follows that an algorithm inverting f_{uni} with probability $\epsilon(n)$, on strings of length $|\mathrm{desc}(M_g)| + n$, yields an algorithm inverting g with probability $\frac{\epsilon(n)}{2^{|\mathrm{desc}(M_g)|}}$ on strings of length n. Hence, if f_{uni} is not weakly one-way then also g cannot be weakly one-way.

Using Theorem 2.8, the proposition follows. ■

The observation, that it suffices to consider one-way functions which can be evaluated within a specific time bound, is crucial to the construction of f_{uni}. The reason being, that it is not possible to construct a polynomial-time machine which is universal for the class of polynomial-time machines (i.e., a polynomial-time machine that can "simulate" all polynomial-time machines). It is however possible to construct, for every polynomial $p(\cdot)$, a polynomial-time machine that is universal for the class of machines with running-time bounded by $p(\cdot)$.

The impracticality of the suggestion to use f_{uni} as a one-way function stems from the fact that f_{uni} is likely to be hard to invert only on huge input lengths.

2.4.2 One-Way Functions as Collections

The formulation of one-way functions, used in so far, is suitable for an abstract discussion. However, for describing many natural candidates for one-way functions, the following formulation (although being more cumbersome) is more adequate. Instead of viewing one-way functions as functions operating on an infinite domain (i.e., $\{0,1\}^*$), we consider infinite collections of of functions each operating on a finite domain. The functions in the collection share a single evaluating algorithm, that given as input a succint representation of a function and an element in its domain, return the value of the specified function at the given point. The formulation of a collection of functions is also useful for the presentation of trapdoor permutations and clawfree functions (see the next two subsections). We start with the following definition.

Definition 2.10 (collection of functions): *A* collection of functions *consists of an infinite set of* indices, *denoted $\overline{I}$, a finite set D_i, for each $i \in \overline{I}$, and a function f_i defined over D_i.*

We will only be interested in collections of functions that can be applied. As hinted above, a necessary condition for applying a collection of functions is the existence of an efficient function-evaluating algorithm (denoted F) that, on input $i \in \overline{I}$ and x, returns $f_i(x)$. Yet, this condition by itself does not suffice. We need to be able to (randomly) select an index, specifying a function over a sufficiently large domain, as well as to be able to (randomly) select an element of the domain (when given the domain's index). The sampling property of the index set is captured by an efficient algorithm (denoted I) that on input an integer n (presented in unary) randomly selects an poly(n)-bit long index, specifying a function and its associated domain. (As usual unary presentation is used to enhence the standard association of efficient algorithms with those running in time polynomial in their length.) The sampling property of the domains is captured by an efficient algorithm (denoted D) that on input an index i randomly selects an element in D_i. The one-way property of the collection is captured by requiring that every efficient algorithm, when given an index of a function and an element in its range, fails to invert the function, except for with negligible probability. The probability is taken over the distribution induced by the sampling algorithms I and D.

Definition 2.11 (collection of one-way functions): *A collection of functions,* $\{f_i : D_i \mapsto \{0,1\}^*\}_{i \in \overline{I}}$, *is called* strongly (resp., weakly) one-way *if there exists three probabilistic polynomial-time algorithms, I, D and F, so that the following two conditions hold*

1. easy to sample and compute: *The output distribution of algorithm I, on input 1^n, is a random variable assigned values in the set $\overline{I} \cap \{0,1\}^n$. The output distribution of algorithm D, on input $i \in \overline{I}$, is a random variable assigned values in D_i. On input $i \in \overline{I}$ and $x \in D_i$, algorithm F always outputs $f_i(x)$.*

2. hard to invert (version for strongly one-way): *For every probabilistic polynomial-time algorithm, A', every polynomial $p(\cdot)$, and all sufficiently large n's*

$$\mathrm{Prob}\left(A'(f_{I_n}(X_n), I_n) \in f_{I_n}^{-1} f_{I_n}(X_n)\right) < \frac{1}{p(n)}$$

   where I_n is a random variable describing the output distribution of algorithm I on input 1^n, and X_n is a random variable describing the output of algorithm D on input (random variable) I_n.
   (The version for weakly one-way collections is analogous.)

We may relate to a collection of one-way functions by indicating the corresponding triplet of algorithms. Hence, we may say that a *triplet of probabilistic polynomial-time algorithms*, (I, D, F), *constitutes a collection of one-way functions* if there exists a collection of functions for which these algorithms satisfy the above two conditions.

We stress that the output of algorithm I, on input 1^n, *is not* necessarily distributed *uniformly* over $\overline{I} \cap \{0,1\}^n$. Furthermore, it is not even required that $I(1^n)$ is not entirely concentrated on one single string. Likewise, the output of algorithm D, on input i, *is not* necessarily distributed *uniformly* over D_i. Yet, the hardness-to-invert condition implies that $D(i)$ cannot be mainly concentrated on polynomially many (in $|i|$) strings. We stress that the collection is hard to invert with respect to the distribution induced by the algorithms I and D (in addition to depending as usual on the mapping induced by the function itself). Clearly, a collection of one-way functions can be represented as a one-way function and vice versa (see Exercise 12), yet each formulation has its advantages. In the sequel we use the formulation of a collection of one-way functions in order to present popular candidates of one-way functions.

To allow less cumbersome presentation of natural candidates of one-way collections (of functions), we relax Definition 2.11 in two ways. First, we allow the index sampling algorithm to output, on input 1^n, indices of length $p(n)$, where $p(\cdot)$ is some polynomial. Secondly, we allow all algorithms to fail with negligible probability. Most importantly, we allow the index sampler I to output strings not in $\overline{I}$ as long as the probability that $I(1^n) \notin \overline{I} \cap \{0,1\}^{p(n)}$ is a negligible function in n. (The same relaxations can be made when discussing trapdoor permutations and clawfree functions.)

2.4.3 Examples of One-way Collections (RSA, Factoring, DLP)

In this subsection we present several popular collections of one-way functions, based on computation number theory (e.g., RSA and Discrete Exponentiation). In the exposition which follows, we assume some knowledge of elementary number theory and some familiarity with simple number theoretic algorithms. Further discussion of the relevant number theoretic material is presented in Appendix `[missing(app-cnt)]`

The RSA function

The RSA collection of functions has an index set consisting of pairs (N,e), where N is a product of two $(\frac{1}{2}\cdot\log_2 N)$-bit primes, denoted P and Q, and e is an integer smaller than N and relatively prime to $(P-1)\cdot(Q-1)$. The function of index (N,e), has domain $\{1,...,N\}$ and maps the domain element x to x^e mod N. Using the fact that e is relatively prime to $(P-1)\cdot(Q-1)$, it can be shown that the fuction is in fact a permutation over its domain. Hence, the RSA collection is a collection of permutations.

We first substantiate the fact that the RSA collection satisfies the first condition of the definition of a one-way collection (i.e., that it is easy to sample and compute). To this end, we present the triplet of algorithms $(I_{\mathrm{RSA}}, D_{\mathrm{RSA}}, F_{\mathrm{RSA}})$.

On input 1^n, algorithm I_{RSA} selects uniformly two primes, P and Q, such that $2^{n-1} \leq P < Q < 2^n$, and an integer e such that e is relatively prime to $(P-1)\cdot(Q-1)$. Algorithm

I_{RSA} terminates with output (N,e), where $N = P \cdot Q$. For an efficient implementation of I_{RSA}, we need a probabilistic polynomial-time algorithms for generating uniformly distributed primes. Such an algorithm does exist. However, it is more efficient to generate two primes by selecting two integers uniformly in the interval $[2^{n-1}, 2^n - 1]$ and checking via a fast randomized primality test whether these are indeed primes (this way we get, with exponentially small probability, an output which is not of the desired form). For more details concerning the uniform generation of primes see Appendix `[missing(app-cnt)]`.

As for algorithm D_{RSA}, on input (N,e), it selects (almost) uniformly an element in the set $D_{N,e} \stackrel{\mathrm{def}}{=} \{1, ..., N\}$. The output of F_{RSA}, on input $((N,e),x)$, is

$$RSA_{N,e}(x) \stackrel{\mathrm{def}}{=} x^e \bmod N$$

It is not known whether factoring N can be reduced to inverting $RSA_{N,e}$, and in fact this is a well-known open problem. We remark that the best algorithms known for inverting $RSA_{N,e}$ proceed by (explicitly or implicitly) factoring N. In any case it is widely believed that the RSA collection is hard to invert.

In the above description $D_{N,e}$ corresponds to the additive group mod N (and hence contain N elements). Alternatively, the domain $D_{N,e}$ can be restricted to the elements of the multiplicative group modulo N (and hence contain $(P-1)\cdot(Q-1) \approx N - 2\sqrt{N} \approx N$ elements). A modified domain sampler may work by selecting an element in $\{1, ..., N\}$ and discarding the unlikely cases in which the selected element is not relatively prime to N. The function $RSA_{N,e}$ defined above indues a permutation on the multiplicative group modulo N. The resulting collection is as hard to invert as the original one. (A proof of this statement is left as an exercise to the reader.) The question which formulation to prefer seems to be a matter of personal taste.

The Rabin function

The Rabin collection of functions is defined analogously to the RSA collection, except that the function is squaring modulo N (instead of raising to the power e mod N). Namely,

$$Rabin_N(x) \stackrel{\mathrm{def}}{=} x^2 \bmod N$$

This function, however, does not induces a permutation on the multiplicative group modulo N, but is rather a 4-to-1 mapping on the multiplicative group modulo N.

It can be shown that extracting square roots modulo N is computationally equivalent to factoring N (i.e., the two tasks are reducible to one another via probabilistic polynomial-time reductions). For details see Exercise 15. Hence, squaring modulo a composite is a collection of one-way functions if and only if factoring is intractable. We remind the reader that it is generally believed that integer factorization is intractable.

The Factoring Permutations

For a special subclass of the integers, known by the name of *Blum Integers*, the function $Rabin_N(\cdot)$ defined above induces a permutation on the quadratic residues modulo N. We say that r is a *quadratic residue mod* N if there exists an integer x such that $r \equiv x^2 \bmod N$. We denote by Q_N the set of quadratic residues in the multiplicative group mod N. For purposes of this paragraph, we say that N is a Blum Integer if it is the product of two primes, each congruent to 3 mod 4. It can be shown that when N is a Blum integer, each element in Q_N has a unique square root which is also in Q_N, and it follows that in this case the function $Rabin_N(\cdot)$ induces a permutation over Q_N. This leads to the introduction of the following collection, $SQR \stackrel{\text{def}}{=} (I_{BI}, D_{QR}, F_{SQR})$, of permutations. On input 1^n, algorithm I_{BI} selects uniformly two primes, P and Q, such that $2^{n-1} \leq P < Q < 2^n$ and $P \equiv Q \equiv 3 \bmod 4$, and outputs $N = P \cdot Q$. It is assumed that the density of such primes is non-negligible and thus that this step can be efficiently implemented. On input N, algorithm D_{QR}, uniformly selects an element of Q_N, by uniformly selecting an element of the multiplicative group modulo N, and squaring it mod N. Algorithm F_{SQR} is defined exactly as in the Rabin collection. The resulting collection is one-way, provided that factoring is intractable also for the set of Blum integers (defined above).

Discrete Logarithms

Another computational number theoretic problem which is widely believed to be intractable is that of extracting discrete logarithms in a finite field (and in particular of prime cardinality). The DLP collection of functions, borrowing its name (and one-wayness) from the *Discrete Logarithm Problem*, is defined by the triplet of algorithms $(I_{\text{DLP}}, D_{\text{DLP}}, F_{\text{DLP}})$.

On input 1^n, algorithm I_{DLP} selects uniformly a prime, P, such that $2^{n-1} \leq P < 2^n$, and a primitive element G in the multiplicative group modulo P (i.e., a generator of this cyclic group), and terminates with output (P, G). There exists a probabilistic polynomial-time algorithm for uniformly generating primes together with the prime factorization of $P-1$, where P is the prime generated (see Appendix `[missing(app-cnt)]`). Alternatively, one may uniformly generate a prime P of the form $2Q+1$, where Q is also a prime. (In the latter case, however, one has to assume the intractability of DLP with respect to such primes. We remark that such primes are commonly believed to be the hardest for DLP.) Using the factorization of $P-1$ one can find a primitive element by selecting an element of the group at random and checking whether it has order $P-1$ (by raising to powers which non-trivially divide $P-1$).

Algorithm D_{DLP}, on input (P, G), selects uniformly a residue modulo $P-1$. Algorithm F_{DLP}, on input $((P, G), x)$, halts outputting

$$DLP_{P,G}(x) \stackrel{\text{def}}{=} G^x \bmod P$$

Hence, inverting $DLP_{P,G}$ amounts to extracting the discrete logarithm (to base G) modulo P. For every (P,G) of the above form, the function $DLP_{P,G}$ induces a 1-1 and onto mapping from the additive group mod $P-1$ to the multiplicative group mod P. Hence, $DLP_{P,G}$ induces a permutation on the the set $\{1,...,P-1\}$.

Exponentiation in other groups is also a reasonable candidate for a one-way function, provided that the discrete logarithm problem for the group is believed to be hard. For example, it is believed that the logarithm problem is hard in the group of points on an Elliptic curve.

Author's Note: *fill-in more details*

2.4.4 Trapdoor one-way permutations

The formulation of collections of one-way functions is convenient as a starting point to the definition of trapdoor permutations. Loosely speaking, these are collections of one-way permutations, $\{f_i\}$, with the extra property that f_i is efficiently inverted once given as auxiliary input a "trapdoor" for the index i. The trapdoor of index i, denoted by $t(i)$, *can not* be efficiently computed from i, yet one can efficiently generate corresponding pairs $(i,t(i))$.

Definition 2.12 (collection of trapdoor permutations): *Let I be a probabilistic algorithm, and let $I_1(1^n)$ (resp. $I_2(1^n)$) denote the first (resp. second) half of the output of $I(1^n)$. A triple of algorithms, (I,D,F), is called a* collection of strong (resp. weak) trapdoor permutations *if the following two conditions hold*

1. the algorithms induce a collection of one-way permutations: *The triple (I_1,D,F) constitutes a collection of one-way permutations.*
2. easy to invert with trapdoor: *There exists a (deterministic) polynomial-time algorithm, denoted F^{-1}, so that for every (i,t) in the range of I and for every $x \in D_i$, it holds that $F^{-1}(t,F(i,x)) = x$.*

A useful relaxation of the above conditions is to require that they are satisfied with overwhelmingly high probability. Namely, the index generating algorithm, I, is allowed to output, with negligible probability, pairs (i,t) for which either f_i is not a permutation or $F^{-1}(t,F(i,x)) = x$ does not hold for all $x \in D_i$.

The RSA (or factoring) Trapdoor

The RSA collection presented above can be easily modified to have the trapdoor property. To this end algorithm I_{RSA} should be modified so that it outputs both the index (N,e) and

the trapdoor (N,d), where d is the multiplicative inverse of e modulo $(P-1)\cdot(Q-1)$ (note that e has such inverse since it has been chosen to be relatively prime to $(P-1)\cdot(Q-1)$). The inverting algorithm F_{RSA}^{-1} is identical to the algorithm F_{RSA} (i.e., $F_{\mathrm{RSA}}^{-1}((N,d),y) = y^d \bmod N$). The reader can easily verify that

$$F_{\mathrm{RSA}}\left((N,d), F_{\mathrm{RSA}}\left((N,e),x\right)\right) = x^{ed} \bmod N$$

indeed equals x for every x in the multiplicative group modulo N. In fact, one can show that $x^{ed} \equiv x \pmod{N}$ for every x (even in case x is not relatively prime to N).

We remark that the Rabin collection presented above can be easily modified in an analogous manner, enabling to efficiently compute all 4 square roots of a given quadratic residue (mod N). The square roots mod N can be computed by extracting a square root modulo each of the primes factors of N and combining the result using the Chinese Reminder Theorem. Efficient algorithms for extracting square root modulo a given prime are known. Furthermore, in case the prime, P, is congruent to 3 mod 4, the square roots of x mod P can be computed by raising x to the power $\frac{P+1}{4}$ (while reducing the intermediate results mod P). Furthermore, in case N is a Blum integer, the collection SQR, presented above, forms a collection of trapdoor permutations (provided of course that factoring is hard).

2.4.5 * Clawfree Functions

Loosely speaking, a clawfree collection consists of a set of pairs of functions which are easy to evaluate, both have the same range, and yet it is infeasible to find a range element together with preimages of it under each of these functions.

Definition 2.13 (clawfree collection): *A collection of pairs of functions consists of an infinite set of indices, denoted $\overline{I}$, two finite sets D_i^0 and D_i^1, for each $i \in \overline{I}$, and two functions f_i^0 and f_i^1 defined over D_i^0 and D_i^1, respectively. Such a collection is called* clawfree *if there exists three probabilistic polynomial-time algorithms, I, D and F, so that the following conditions hold*

1. easy to sample and compute: *The random variable $I(1^n)$ is assigned values in the set $\overline{I} \cap \{0,1\}^n$. For each $i \in \overline{I}$ and $\sigma \in \{0,1\}$, the random variable $D(\sigma,i)$ is distributed over D_i^σ and $F(\sigma,i,x) = f_i^\sigma(x)$.*
2. identical range distribution: *For every i in the index set $\overline{I}$, the random variables $f_i^0(D(0,i))$ and $f_i^1(D(1,i))$ are identically distributed.*
3. hard to form claws: *A pair (x,y) satisfying $f_i^0(x) = f_i^1(y)$ is called a* claw *for index i. Let C_i denote the set of claws for index i. It is required that for every probabilistic polynomial-time algorithm, A', every polynomial $p(\cdot)$, and all sufficiently large n's*

$$\mathrm{Prob}\left(A'(I_n) \in C_{I_n}\right) < \frac{1}{p(n)}$$

where I_n is a random variable describing the output distribution of algorithm I on input 1^n.

The first requirement in Definition 2.13 is analogous to what appears in Definition 2.11. The other two requirements (in Definition 2.13) are kind of conflicting. On one hand, it is required that that claws do exist (to say the least), whereas on the other hand it is required that calws cannot be efficiently found. Clearly, a clawfree collection of functions yields a collection of strong one-way functions (see Exercise 16). A special case of interest is when both domains are identical (i.e., $D_i \stackrel{\text{def}}{=} D_i^0 = D_i^1$), the random variable $D(\sigma, i)$ is uniformly distributed over D_i, and the functions, f_i^0 and f_i^1, are permutations over D_i. Such a collection is called a collection of (clawfree) permutations.

Again, a useful relaxation of the conditions of Definition 2.13 is obtained by allowing the algorithms (i.e., I, D and F) to fail with negligible probability.

An additional property that a (clawfree) collection may (or may not) have is an efficiently recognizable index set (i.e., an probabilistic polynomial-time algorithm for determining whether a give string is $\overline{I}$). This property is useful in some applications of clawfree collections (hence this discussion). Efficient recognition of the index set may be important since the function-evaluating algorithm F may induce functions also in case its second input (which is supposedly an index) is not in $\overline{I}$. In this case it is no longer guaranteed that the induced pair of functions has identical range distribution. In some applications (e.g., see section 6.8), dishonest parties may choose, on purpose, an illegal index and try to capitalize on the induce functions having different range distributions.

The DLP Clawfree Collection

We now turn to show that clawfree collections do exists under specific reasonable intractability assumptions. We start by presenting such a collection under the assumption that the Discrete Logarithm Problem (DLP) for fields of prime cardinality is intractable.

Following is the description a collection of clawfree *permutations* (based on the above assumption). The index sets consists of triples, $P, G, Z)$, where P is a prime, G is a primitive element mod P, and Z is an element in the field (of residues mod P). The index sampling algorithm, selects P and G as in the DLP collection presented in Subsection 2.4.3, and Z is selected uniformly among the residues mod P. The domain of both functions with index (P, G, Z) is identical, and equals the set $\{1, ..., P-1\}$, and the domain sampling algorithm selects uniformly from this set. As for the functions themselves, we set

$$f^{\sigma}_{P,G,Z}(x) \stackrel{\text{def}}{=} Z^{\sigma} \cdot G^{x} \bmod P$$

The reader can easily verify that both functions are permutations over $\{1, ..., P-1\}$. Also, the ability to form a claw for the index (P, G, Z) yields the ability to find the discrete

logarithm of Z mod P to base G (since $G^x \equiv Z \cdot G^y \bmod P$ yields $G^{x-y} \equiv Z \bmod P$). Hence, ability to form claws for a non-negligible fraction of the index set translates to a contradiction to the DLP intractability assumption.

The above collection *does not* have the additional property of having an efficiently recognizable index set, since it is not known how to efficiently recognize primitive elements modulo a prime. This can be amended by making a slightly stronger assumption concerning the intractability of DLP. Specifically, we assume that DLP is intractable even if one is given the factorization of the size of the multiplicative group (i.e., the factorization of $P-1$) as additional input. Such an assumption allows to add the factorization of $P-1$ into the description of the index. This makes the index set efficiently recognizable (since one can first test P for primality, as usual, and next test whether G is a primitive element by raising it to powers of the form $(P-1)/Q$ where Q is a prime factor of $P-1$). If DLP is hard also for primes of the form $2Q+1$, where Q is also a prime, life is even easier. To test whether G is a primitive element mod P one just computes $G^2 \pmod P$ and $G^{(P-1)/2} \pmod P$, and checks whether either of them equals 1.

The Factoring Clawfree Collection

We now show that a clawfree collection (of functions) does exists under the assumption that integer factorization is infeasible for integers which are the product of two primes each congruent to 3 mod 4. Such composite numbers, hereafter referred to as *Blum integers*, have the property that the Jacobi symbol of -1 (relative to them) is 1 and half of the square roots of each quadratic residue, in the corresponding multiplicative group (modulo this composite), have Jacobi symbol 1 (see Appendix `[missing(app-cnt)]`).

The index set of the collection consists of all Blum integers which are composed of two primes of equal length. The index selecting algorithm, on input 1^n, uniformly select such an integers, by uniformly selecting two (n-bit) primes each congruent to 3 mod 4, and outputting their product, denoted N. Let J_N^{+1} (respectively, J_N^{-1}) denote the set of residues in the multiplicative group modulo N with Jacobi Symbol $+1$ (resp., -1). The functions of index N, denoted f_N^0 and f_N^1, consist both of squaring modulo N, but their corresponding domains are disjoint. The domain of function f_N^σ equals the set $J_N^{(-1)^\sigma}$. The domain sampling algorithm, denoted D, uniformly selects an element of the corresponding domain as follows. Specifically, on input (σ, N) algorithm D uniformly selects polynomially many residues mod N, and outputs the first residue with Jacobi Symbol $(-1)^\sigma$.

The reader can easily verify that both $f_N^0(D(0,N))$ and $f_N^1(D(1,N))$ are uniformly distributed over the set of quadratic residues mod N. The difficult of forming claws follows from the fact that a claw yield two residues, $x \in J_N^{+1}$ and $y \in J_N^{-1}$ such that $x^2 \equiv y^2 \pmod N$. Since $-1 \in J_N^{+1}$, it follows that $x \neq \pm y$ and the gcd of $x \pm y$ and N yields a factorization of N.

The above collection *does not* have the additional property of having an efficiently recognizable index set, since it is not even known how to efficiently distinguish products of two primes from products of more than two primes.

2.4.6 On Proposing Candidates

Although we do believe that one-way functions exist, their *mere* existence does not suffice for practical applications. Typically, an application which is based on one-way functions requires the specification of a concrete (candidate one-way) function. As explained above, the observation concerning the existence of a universal one-way function is of little practical significance. Hence, the problem of proposing reasonable candidates for one-way functions is of great practical importance. Everyone understands that such a reasonable candidate (for a one-way function) should have a very efficient algorithm for evaluating the function. (In case the "function" is presented as a collection of one-way functions, especially the domain sampler and function-evaluation algorithm should be very efficient.) However, people seem less careful in *seriously considering* the difficulty of inverting the candidates that they propose. We stress that the candidate has to be difficult to invert on "the average" and not only on the worst case, and that "the average" is taken with respect to the instance-distribution determined by the candidate function. Furthermore, "hardness on the average" (unlike worst case analysis) is extremely sensitive to the instance-distribution. Hence, one has to be extremely careful in deducing average-case complexity with respect to one distribution from the average-case complexity with respect to another distribution. The short history of the field contains several cases in which this point has been ignored and consequently bad suggestions has been made.

Consider for example the following suggestion to base one-way functions on the conjectured difficulty of the Graph Isomorphism problem. Let $f_{\mathrm{GI}}(G,\pi) = (G,\pi G)$, where G is an undirected graph, π is a permutation on its vertex set, and πG denotes the graph resulting by renaming the vertices of G using π (i.e., $(\pi(u),\pi(v))$ is an edge in πG iff (u,v) is an edge in G). Although it is indeed believed that Graph Isomorphism cannot be solved in polynomial-time, it is easy to see that F_{GI} is easy to invert on most instances (e.g., use vertex degree statistics to determine the isomorphism).

2.5 Hard-Core Predicates

Loosely speaking, saying that a function f is one-way means that given y it is infeasible to find a preimage of y under f. This does not mean that it is infeasible to find out partial information about the preimage of y under f. Specifically it may be easy to retrieve half of the bits of the preimage (e.g., given a one-way function f consider the function g defined by $g(x,r) \stackrel{\mathrm{def}}{=} (f(x),r)$, for every $|x| = |r|$). The fact that one-way functions do not

necessarily hide partial information about their preimage limits their "direct applicability" to tasks as secure encryption. Fortunately, assuming the existence of one-way functions, it is possible to construct one-way functions which hide specific partial information about their preimage (which is easy to compute from the preimage itself). This partial information can be considered as a "hard core" of the difficulty of inverting f.

2.5.1 Definition

A *polynomial-time* predicate b, is called a hard-core of a function f if all efficient algorithm, given $f(x)$, can guess $b(x)$ only with success probability which is negligibly better than half.

Definition 2.14 (hard-core predicate): *A polynomial-time computable predicate $b : \{0,1\}^* \mapsto \{0,1\}$ is called a* hard-core *of a function f if for every probabilistic polynomial-time algorithm A', every polynomial $p(\cdot)$, and all sufficiently large n's*

$$\text{Prob}\left(A'(f(U_n)) = b(U_n)\right) < \frac{1}{2} + \frac{1}{p(n)}$$

It follows that if b is a hard-core predicate (for any function) then $b(U_n)$ should be almost unbiased (i.e., $|\text{Prob}(b(U_n)=0) - \text{Prob}(b(U_n)=1)|$ must be a negligible function in n). As b itself is polynomial-time computable the failure of efficient algorithms to approximate $b(x)$ from $f(x)$ (with success probability significantly more than half) must be due to either an information loss of f (i.e., f not being one-to-one) or to the difficulty of inverting f. For example, the predicate $b(\sigma\alpha) = \sigma$ is a hard-core of the function $f(\sigma\alpha) \stackrel{\text{def}}{=} 0\alpha$, where $\sigma \in \{0,1\}$ and $\alpha \in \{0,1\}^*$. Hence, in this case the fact that b is a hard-core of the function f is due to the fact that f losses information (specifically the first bit σ). On the other hand, in case f losses no information (i.e., f is one-to-one) hard-cores for f exist only if f is one-way (see Exercise 19). Finally, we note that for every b and f, there exist obvious algorithms which guess $b(U_n)$ from $f(U_n)$ with success probability at least half (e.g., either an algorithm A_1 that regardless of its input answers with a uniformly chosen bit, or, in case b is not biased towards 0, the constant algorithm $A_2(x) \stackrel{\text{def}}{=} 1$).

Simple hard-core predicates are known for the RSA, Rabin, and DLP collections (presented in Subsection 2.4.3), provided that the corresponding collections are one-way. Specifically, the least significant bit is a hard-core for the RSA collection, provided that the RSA collection is one-way. Namely, assuming that the RSA collection is one-way, it is infeasible to guess (with success probability significantly greater than half) the least significant bit of x from $RSA_{N,e}(x) = x^e \bmod N$. Likewise, assuming that the DLP collection is one-way, it is infeasible to guess whether $x < \frac{P}{2}$ when given $DLP_{P,G}(x) = G^x \bmod P$. In the next subsection we present a general result of the kind.

2.5.2 Hard-Core Predicates for any One-Way Function

Actually, the title is inaccurate, as we are going to present hard-core predicates only for (strong) one-way functions of special form. However, every (strong) one-way function can be easily transformed into a function of the required form, with no substantial loss in either "security" or "efficiency".

Theorem 2.15 *Let f be an arbitrary strong one-way function, and let g be defined by $g(x,r) \stackrel{\text{def}}{=} (f(x),r)$, where $|x|=|r|$. Let $b(x,r)$ denote the inner-product mod 2 of the binary vectors x and r. Then the predicate b is a hard-core of the function g.*

In other words, the theorem states that if f is strongly one-way then it is infeasible to guess the exclusive-or of a random subset of the bits of x when given $f(x)$ and the subset itself. We stress that the theorem requires that f is strongly one-way and that the conclusion is false if f is only weakly one-way (see Exercise 19). We point out that g maintains properties of f such as being length-preserving and being one-to-one. Furthermore, an analogous statement holds for collections of one-way functions with/without trapdoor etc.

Proof: The proof uses a "reducibility argument". This time inverting the function f is reduced to predicting $b(x,r)$ from $(f(x),r)$. Hence, we assume (for contradiction) the existence of an efficient algorithm predicting the inner-product with advantage which is not negligible, and derive an algorithm that inverts f with related (i.e. not negligible) success probability. This contradicts the hypothesis that f is a one-way function.

Let G be a (probabilistic polynomial-time) algorithm that on input $f(x)$ and r tries to predict the inner-product (mod 2) of x and r. Denote by $\varepsilon_G(n)$ the (overall) advantage of algorithm G in predicting $b(x,r)$ from $f(x)$ and r, where x and r are uniformly chosen in $\{0,1\}^n$. Namely,

$$\varepsilon_G(n) \stackrel{\text{def}}{=} \text{Prob}\left(G(f(X_n),R_n) = b(X_n,R_n)\right) - \frac{1}{2}$$

where here and in the sequel X_n and R_n denote two independent random variables, each uniformly distributed over $\{0,1\}^n$. Assuming, to the contradiction, that b is not a hard-core of g means that exists an efficient algorithm G, a polynomial $p(\cdot)$ and an infinite set N so that for every $n \in N$ it holds that $\varepsilon_G(n) > \frac{1}{p(n)}$. We restrict our attention to this algorithm G and to n's in this set N. In the sequel we shorthand ε_G by ε.

Our first observation is that, on at least an $\frac{\varepsilon(n)}{2}$ fraction of the x's of length n, algorithm G has an $\frac{\varepsilon(n)}{2}$ advantage in predicting $b(x,R_n)$ from $f(x)$ and R_n. Namely,

Claim 2.15.1: there exists a set $S_n \subseteq \{0,1\}^n$ of cardinality at least $\frac{\varepsilon(n)}{2} \cdot 2^n$ such that for every $x \in S_n$, it holds that

$$s(x) \stackrel{\text{def}}{=} \text{Prob}(G(f(x),R_n)=b(x,R_n)) \geq \frac{1}{2} + \frac{\varepsilon(n)}{2}$$

This time the probability is taken over all possible values of R_n and all internal coin tosses of algorithm G, whereas x is fixed.

Proof: The observation follows by an averaging argument. Namely, write $\mathrm{Exp}(s(X_n)) = \frac{1}{2} + \varepsilon(n)$, and apply Markov Inequality.□

In the sequel we restrict our attention to x's in S_n. We will show an efficient algorithm that on every input y, with $y = f(x)$ and $x \in S_n$, finds x with very high probability. Contradiction to the (strong) one-wayness of f will follow by noting that $\mathrm{Prob}(U_n \in S_n) \geq \frac{\varepsilon(n)}{2}$.

The next three paragraphs consist of a motivating discussion. The inverting algorithm, that uses algorithm G as subroutine, will be formally described and analyzed later.

A motivating discussion

Consider a fixed $x \in S_n$. By definition $s(x) \geq \frac{1}{2} + \frac{\varepsilon(n)}{2} > \frac{1}{2} + \frac{1}{2p(n)}$. Suppose, for a moment, that $s(x) > \frac{3}{4} + \frac{1}{2p(n)}$. Of course there is no reason to believe that this is the case, we are just doing a mental experiment. In this case (i.e., of $s(x) > \frac{3}{4} + \frac{1}{\mathrm{poly}(|x|)}$) retrieving x from $f(x)$ is quite easy. To retrieve the i^{th} bit of x, denoted x_i, we randomly select $r \in \{0,1\}^n$, and compute $G(f(x), r)$ and $G(f(x), r \oplus e^i)$, where e^i is an n-dimensional binary vector with 1 in the i^{th} component and 0 in all the others, and $v \oplus u$ denotes the addition mod 2 of the binary vectors v and u. Clearly, if both $G(f(x), r) = b(x, r)$ and $G(f(x), r \oplus e^i) = b(x, r \oplus e^i)$, then

$$\begin{aligned} G(f(x), r) \oplus G(f(x), r \oplus e^i) &= b(x, r) \oplus b(x, r \oplus e^i) \\ &= b(x, e^i) \\ &= x_i \end{aligned}$$

since $b(x, r) \oplus b(x, s) \equiv \sum_{i=1}^{n} x_i r_i + \sum_{i=1}^{n} x_i s_i \equiv \sum_{i=1}^{n} x_i (r_i + s_i) \equiv b(x, r \oplus s) \bmod 2$. The probability that both equalities hold (i.e., both $G(f(x), r) = b(x, r)$ and $G(f(x), r \oplus e^i) = b(x, r \oplus e^i)$) is at least $1 - 2 \cdot (\frac{1}{4} - \frac{1}{\mathrm{poly}(|x|)}) > 1 - \frac{1}{\mathrm{poly}(|x|)}$. Hence, repeating the above procedure sufficiently many times and ruling by majority we retrieve x_i with very high probability. Similarly, we can retrieve all the bits of x, and hence invert f on $f(x)$. However, the entire analysis was conducted under (the unjustifiable) assumption that $s(x) > \frac{3}{4} + \frac{1}{2p(|x|)}$, whereas we only know that $s(x) > \frac{1}{2} + \frac{1}{2p(|x|)}$.

The problem with the above procedure is that it doubles the original error probability of algorithm G on inputs of form $(f(x), \cdot)$. Under the unrealistic assumption, that the G's error on such inputs is significantly smaller than $\frac{1}{4}$, the "error-doubling" phenomenon raises no problems. However, in general (and even in the special case where G's error is exactly $\frac{1}{4}$) the above procedure is unlikely to invert f. Note that the error probability of G can not be decreased by repeating G several times (e.g., G may always answer correctly on three quarters of the inputs, and always err on the remaining quarter). What is required

is an *alternative way of using* the algorithm G, a way which does not double the original error probability of G. The key idea is to generate the r's in a way which requires applying algorithm G only once per each r (and i), instead of twice. Specifically, we used algorithm G to obtain a "guess" for $b(x, r \oplus e^i)$ and obtain $b(x, r)$ in a different way. The good news are that the error probability is no longer doubled, since we only need to use G to get a "guess" of $b(x, r \oplus e^i)$. The bad news are that we still need to know $b(x, r)$, and it is not clear how we can know $b(x, r)$ without applying G. The answer is that we can guess $b(x, r)$ by ourselves. This is fine if we only need to guess $b(x, r)$ for one r (or logarithmically in $|x|$ many r's), but the problem is that we need to know (and hence guess) $b(x, r)$ for polynomially many r's. An obvious way of guessing these $b(x, r)$'s yields an exponentially vanishing success probability. The solution is to generate these polynomially many r's so that, on one hand they are "sufficiently random" whereas on the other hand we can guess all the $b(x, r)$'s with non-negligible success probability. Specifically, generating the r's in a particular *pairwise independent* manner will satisfy both (seemingly contradictory) requirements. We stress that in case we are successful (in our guesses for the $b(x, r)$'s), we can retrieve x with high probability. Hence, we retrieve x with non-negligible probability.

A word about the way in which the pairwise independent r's are generated (and the corresponding $b(x, r)$'s are guessed) is indeed in place. To generate $m = \mathrm{poly}(n)$ many r's, we uniformly (and independently) select $l \stackrel{\mathrm{def}}{=} \log_2(m+1)$ strings in $\{0,1\}^n$. Let us denote these strings by $s^1, ..., s^l$. We then guess $b(x, s^1)$ through $b(x, s^l)$. Let use denote these guesses, which are uniformly (and independently) chosen in $\{0,1\}$, by σ^1 through σ^l. Hence, the probability that all our guesses for the $b(x, s^i)$'s are correct is $2^{-l} = \frac{1}{\mathrm{poly}(n)}$. The different r's correspond to the different non-empty subsets of $\{1, 2, ..., l\}$. We compute $r^J \stackrel{\mathrm{def}}{=} \oplus_{j \in J} s^j$. The reader can easily verify that the r^J's are pairwise independent and each is uniformly distributed in $\{0,1\}^n$. The key observation is that

$$b(x, r^J) = b(x, \oplus_{j \in J} s^j) = \oplus_{j \in J} b(x, s^j)$$

Hence, our guess for the $b(x, r^J)$'s is $\oplus_{j \in J} \sigma^j$, and with non-negligible probability all our guesses are correct.

Back to the formal argument

Following is a formal description of the inverting algorithm, denoted A. We assume, for simplicity that f is length preserving (yet this assumption is not essential). On input y (supposedly in the range of f), algorithm A sets $n \stackrel{\mathrm{def}}{=} |y|$, and $l \stackrel{\mathrm{def}}{=} \lceil \log_2(2n \cdot p(n)^2 + 1) \rceil$, where $p(\cdot)$ is the polynomial guaranteed above (i.e., $\epsilon(n) > \frac{1}{p(n)}$ for the infinitely many n's in N). Algorithm A uniformly and independently select $s^1, ..., s^l \in \{0,1\}^n$, and $\sigma^1, ..., \sigma^l \in \{0,1\}$. It then computes, for every non-empty set $J \subseteq \{1, 2, ..., l\}$, a string $r^J \leftarrow \oplus_{j \in J} s^j$ and a bit $\rho^J \leftarrow \oplus_{j \in J} \sigma^j$. For every $i \in \{1, ..., n\}$ and every *non-empty* $J \subseteq \{1, .., l\}$, algorithm A computes $z_i^J \leftarrow \rho^J \oplus G(y, r^J \oplus e^i)$. Finally, algorithm A sets z_i to be the majority of the z_i^J

values, and outputs $z = z_1 \cdots z_n$. (Remark: in an alternative implementation of the ideas, the inverting algorithm, denoted A', tries all possible values for $\sigma^1, ..., \sigma^l$, and outputs only one of resulting strings z, with an obvious preference to a string z satisfying $f(z) = y$.)

Following is a detailed analysis of the success probability of algorithm A on inputs of the form $f(x)$, for $x \in S_n$, where $n \in N$. We start by showing that, in case the σ^j's are correct, then the with constant probability, $z_i = x_i$ for all $i \in \{1, ..., n\}$. This is proven by bounding from below the probability that the majority of the z_i^J's equals x_i.

Claim 2.15.2: For every $x \in S_n$ and every $1 \leq i \leq n$,

$$\text{Prob}\left(|\{J : b(x, r^J) \oplus G(f(x), r^J \oplus e^i) = x_i\}| > \frac{1}{2} \cdot (2^l - 1)\right) > 1 - \frac{1}{2n}$$

where $r^J \stackrel{\text{def}}{=} \oplus_{j \in J} s^j$ and the s^j's are independently and uniformly chosen in $\{0,1\}^n$.

Proof: For every J, define a 0-1 random variable ζ^J, so that ζ^J equals 1 if and only if $b(x, r^J) \oplus G(f(x), r^J \oplus e^i) = x_i$. The reader can easily verify that each r^J is uniformly distributed in $\{0,1\}^n$. It follows that each ζ^J equals 1 with probability $s(x)$, which by $x \in S_n$, is at least $\frac{1}{2} + \frac{1}{2p(n)}$. We show that the ζ^J's are pairwise independent by showing that the r^J's are pairwise independent. For every $J \neq K$ we have, without loss of generality, $j \in J$ and $k \in K - J$. Hence, for every $\alpha, \beta \in \{0,1\}^n$, we have

$$\begin{aligned} \text{Prob}\left(r^K = \beta \mid r^J = \alpha\right) &= \text{Prob}\left(s^k = \beta \mid s^j = \alpha\right) \\ &= \text{Prob}\left(s^k = \beta\right) \\ &= \text{Prob}\left(r^K = \beta\right) \end{aligned}$$

and pairwise independence of the r^J's follows. Let $m \stackrel{\text{def}}{=} 2^l - 1$. Using Chebyshev's Inequality, we get

$$\begin{aligned} \text{Prob}\left(\sum_J \zeta^J \leq \frac{1}{2} \cdot m\right) &\leq \text{Prob}\left(|\sum_J \zeta^J - (\frac{1}{2} + \frac{1}{2p(n)}) \cdot m| \geq \frac{1}{2p(n)} \cdot m\right) \\ &< \frac{\text{Var}(\zeta^{\{1\}})}{(\frac{1}{2p(n)})^2 \cdot (2n \cdot p(n)^2)} \\ &< \frac{\frac{1}{4}}{(\frac{1}{2p(n)})^2 \cdot (2n \cdot p(n)^2)} \\ &= \frac{1}{2n} \end{aligned}$$

The claim now follows. □

Recall that if $\sigma^j = b(x, s^j)$, for all j's, then $\rho^J = b(x, r^J)$ for all non-empty J's. In this case z output by algorithm A equals x, with probability at least half. However, the first event happens with probability $2^{-l} = \frac{1}{2n \cdot p(n)^2}$ independently of the events analyzed in Claim 2.15.2. Hence, in case $x \in S_n$, algorithm A inverts f on $f(x)$ with probability at least $\frac{1}{4p(|x|)}$ (whereas, the modified algorithm, A', succeeds with probability $\geq \frac{1}{2}$). Recalling that $|S_n| > \frac{1}{2p(n)} \cdot 2^n$, we conclude that, for every $n \in N$, algorithm A inverts f on $f(U_n)$ with probability at least $\frac{1}{8p(n)^2}$. Noting that A is polynomial-time (i.e., it merely invokes G for $2n \cdot p(n)^2 = \text{poly}(n)$ times in addition to making a polynomial amount of other computations), a contradiction, to our hypothesis that f is strongly one-way, follows. ■

2.5.3 * Hard-Core Functions

We have just seen that every one-way function can be easily modified to have a hard-core predicate. In other words, the result establishes one bit of information about the preimage which is hard to approximate from the value of the function. A stronger result may say that several bits of information about the preimage are hard to approximate. For example, we may want to say that a specific pair of bits is hard to approximate, in the sense that it is infeasible to guess this pair with probability significantly larger than $\frac{1}{4}$. In general, a *polynomial-time* function, h, is called a hard-core of a function f if no efficient algorithm can distinguish $(f(x), h(x))$ from $(f(x), r)$, where r is a random string of length $|h(x)|$. For further discussion of the notion of efficient distinguishability the reader is referred to Section 3.2. We assume for simplicity that h is length regular (see below).

Definition 2.16 (hard-core function): *Let $h : \{0,1\}^* \mapsto \{0,1\}^*$ be a polynomial-time computable function, satisfying $|h(x)| = |h(y)|$ for all $|x| = |y|$, and let $l(n) \stackrel{\text{def}}{=} |h(1^n)|$. The function $h : \{0,1\}^* \mapsto \{0,1\}^*$ is called a* hard-core *of a function f if for every probabilistic polynomial-time algorithm D', every polynomial $p(\cdot)$, and all sufficiently large n's*

$$|\text{Prob}\left(D'(f(X_n), h(X_n)) = 1\right) - \text{Prob}\left(D'(f(X_n), R_{l(n)}) = 1\right)| < \frac{1}{p(n)}$$

where X_n and $R_{l(n)}$ are two independent random variables the first uniformly distributed over $\{0,1\}^n$, and the second uniformly distributed over $\{0,1\}^{l(n)}$,

Theorem 2.17 *Let f be an arbitrary strong one-way function, and let g_2 be defined by $g_2(x,s) \stackrel{\text{def}}{=} (f(x), s)$, where $|s| = 2|x|$. Let $c > 0$ be a constant, and $l(n) \stackrel{\text{def}}{=} \lceil c \log_2 n \rceil$. Let $b_i(x,s)$ denote the inner-product mod 2 of the binary vectors x and $(s_{i+1}, ..., s_{i+n})$, where $s = (s_1, ..., s_{2n})$. Then the function $h(x,s) \stackrel{\text{def}}{=} b_1(x,s) \cdots b_{l(|x|)}(x,s)$ is a hard-core of the function g_2.*

The proof of the theorem follows by combining a proposition concerning the structure of the specific function h with a general lemma concerning hard-core functions. Loosely speaking, the proposition "reduces" the problem of approximating $b(x,r)$ given $g(x,r)$ to the problem of approximating the exclusive-or of any non-empty set of the bits of $h(x,s)$ given $g_2(x,s)$, where b and g are the hard-core and the one-way function presented in the previous subsection. Since we know that the predicate $b(x,r)$ cannot be approximated from $g(x,r)$, we conclude that no exclusive-or of the bits of $h(x,s)$ can be approximated from $g_2(x,s)$. The general lemma states that, for every "logarithmically shrinking" function h' (i.e., h' satisfying $|h'(x)| = O(\log |x|)$), the function h' is a hard-core of a function f' if and only if the exclusive-or of any non-empty subset of the bits of h' cannot be approximated from the value of f'.

Proposition 2.18 *Let f, g_2 and b_i's be as above. Let $I(n) \subseteq \{1,2,...,l(n)\}$, $n \in \mathbb{N}$, be an arbitrary sequence of non-empty subsets, and let $b_{I(|x|)}(x,s) \stackrel{\text{def}}{=} \oplus_{i \in I(|x|)} b_i(x,s)$. Then, for every probabilistic polynomial-time algorithm A', every polynomial $p(\cdot)$, and all sufficiently large n's*

$$\text{Prob}\left(A'(g_2(U_{3n})) = b_{I(n)}(U_{3n})\right) < \frac{1}{2} + \frac{1}{p(n)}$$

Proof: The proof is by a "reducibility" argument. It is shown that the problem of approximating $b(X_n, R_n)$ given $(f(X_n), R_n)$ is reducible to the problem of approximating $b_{I(n)}(X_n, S_{2n})$ given $(f(X_n), S_{2n})$, where X_n, R_n and S_{2n} are independent random variable and the last is uniformly distributed over $\{0,1\}^{2n}$. The underlying observation is that, for every $|s| = 2 \cdot |x|$,

$$b_I(x,s) = \oplus_{i \in I} b_i(x,s) = b(x, \oplus_{i \in I} \text{sub}_i(s)$$

where $\text{sub}_i(s_1,...,s_{2n}) \stackrel{\text{def}}{=} (s_{i+1},...,s_{i+n})$. Furthermore, the reader can verify that for every non-empty $I \subseteq \{1,...,n\}$, the random variable $\oplus_{i \in I} \text{sub}_i(S_{2n})$ is uniformly distributed over $\{0,1\}^n$, and that given a string $r \in \{0,1\}^n$ and such a set I one can efficiently select a string uniformly in the set $\{s : \oplus_{i \in I} \text{sub}_i(s) = r\}$. (Verification of both claims is left as an exercise.)

Now, assume to the contradiction, that there exists an efficient algorithm A', a polynomial $p(\cdot)$, and an infinite sequence of sets (i.e., $I(n)$'s) and n's so that

$$\text{Prob}\left(A'(g_2(U_{3n})) = b_{I(n)}(U_{3n})\right) \geq \frac{1}{2} + \frac{1}{p(n)}$$

We first observe that for n's satisfying the above inequality we can find in probabilistic polynomial time (in n) a set I satisfying

$$\text{Prob}\left(A'(g_2(U_{3n})) = b_I(U_{3n})\right) \geq \frac{1}{2} + \frac{1}{2p(n)}$$

(i.e., by going over all possible I's and experimenting with algorithm A' on each of them). Of course we may be wrong here, but the error probability can be made exponentially small.

We now present an algorithm for approximating $b(x,r)$, from $y \stackrel{\text{def}}{=} f(x)$ and r. On input y and r, the algorithm first finds a set I as described above (this stage depends only on $|x|$ which equals $|r|$). Once I is found, the algorithm uniformly select a string s so that $\oplus_{i\in I}\text{sub}_i(s) = r$, and return $A'(y,s)$. Evaluation of the success probability of this algorithm is left as an exercise. ■

Lemma 2.19 (Computational XOR Lemma): *Let f and h be arbitrary length regular functions, and let $l(n) \stackrel{\text{def}}{=} |h(1^n)|$. Let D be an algorithm. Denote*

$$p \stackrel{\text{def}}{=} \text{Prob}\left(D(f(X_n),h(X_n))=1\right) \quad \textit{and} \quad q \stackrel{\text{def}}{=} \text{Prob}\left(D(f(X_n),R_{l(n)})=1\right)$$

where X_n and R_l are as above. Let G be an algorithm that on input y, S (and $l(n)$), selects r uniformly in $\{0,1\}^{l(n)}$, and outputs $D(y,r)\oplus 1\oplus(\oplus_{i\in S}r_i)$, where $r = r_1\cdots r_l$ and $r_i \in \{0,1\}$. Then,

$$\text{Prob}\left(G(f(X_n),I_l,l(n)) = \oplus_{i\in I_l}(h_i(X_n))\right) = \frac{1}{2} + \frac{p-q}{2^{l(n)}-1}$$

where I_l is a randomly chosen non-empty subset of $\{1,...,l(n)\}$ and $h_i(x)$ denotes the i^{th} bit of $h(x)$.

It follows that, for logarithmically shrinking h's, the existence of an efficient algorithm that distinguishes (with a gap which is not negligible in n) the random variables $(f(X_n),h(X_n))$ and $(f(X_n),R_{l(n)})$ implies the existence of an efficient algorithm that approximates the exclusive-or of a random non-empty subset of the bits of $h(X_n)$ from the value of $f(X_n)$ with an advantage that is not negligible. On the other hand, it is clear that any efficient algorithm, which approximates an exclusive-or of an non-empty subset of the bits of h from the value of f, can be easily modified to distinguish $(f(X_n),h(X_n))$ from $(f(X_n),R_{l(n)})$. Hence, for logarithmically shrinking h's, the function h is a hard-core of a function f if and only if the exclusive-or of any non-empty subset of the bits of h cannot be approximated from the value of f.

Proof: All that is required is to evaluate the success probability of algorithm G. We start by fixing an $x \in \{0,1\}^n$ and evaluating $\text{Prob}(G(f(x),I_l,l) = \oplus_{i\in I_l}(h_i(x))$, where I_l is a uniformly chosen non-empty subset of $\{1,...,l\}$ and $l \stackrel{\text{def}}{=} l(n)$. Let B denote the set of all non-empty subsets of $\{1,...,l\}$. Define, for every $S \in B$, a relation $\equiv_S$ so that $y \equiv_S z$ if and only if $\oplus_{i\in S}y_i = \oplus_{i\in S}z_i$, where $y = y_1\cdots y_l$ and $z = z_1\cdots z_l$. By the definition of G, it follows that on input $(f(x),S,l)$ and random choice $r \in \{0,1\}^l$, algorithm G outputs $\oplus_{i\in S}(h_i(x))$

if and only if either "$D(f(x),r)=1$ and $r \equiv_S h(x)$" or "$D(f(x),r)=0$ and $r \not\equiv_S h(x)$". By elementary manipulations, we get

$$\begin{aligned}
s(x) &\stackrel{\text{def}}{=} \mathrm{Prob}(G(f(x),I_l,l) = \oplus_{i\in I_l}(h_i(x))) \\
&= \sum_{S\in B} \frac{1}{|B|}\mathrm{Prob}(G(f(x),S,l) = \oplus_{i\in S}(h_i(x)) \\
&= \sum_{S\in B} \frac{1}{2\cdot|B|}\left(\mathrm{Prob}(D(f(x),R_l)=1 \mid R_l \equiv_S h(x)) + \mathrm{Prob}(D(f(x),R_l)=0 \mid R_l \not\equiv_S h(x))\right) \\
&= \frac{1}{2} + \frac{1}{2|B|}\sum_{S\in B}\left(\mathrm{Prob}(D(f(x),R_l)=1 \mid R_l \equiv_S h(x)) - \mathrm{Prob}(D(f(x),R_l)=1 \mid R_l \not\equiv_S h(x))\right) \\
&= \frac{1}{2} + \frac{1}{2|B|}\cdot\frac{1}{2^{l-1}}\cdot\left(\sum_{S\in B}\sum_{r\equiv_S h(x)}\mathrm{Prob}(D(f(x),r)=1) - \sum_{S\in B}\sum_{r\not\equiv_S h(x)}\mathrm{Prob}(D(f(x),x)=1)\right) \\
&= \frac{1}{2} + \frac{1}{2^l\cdot|B|}\cdot\left(\sum_{r}\sum_{S\in E(r,h(x))}\mathrm{Prob}(D(f(x),r)=1) - \sum_{r}\sum_{S\in N(r,h(x))}\mathrm{Prob}(D(f(x),r)=1)\right)
\end{aligned}$$

where $E(r,z) \stackrel{\text{def}}{=} \{S \in B : r \equiv_S z\}$ and $N(r,z) \stackrel{\text{def}}{=} \{S \in B : r \not\equiv_S z\}$. Observe that for every $r \neq z$ it holds that $|N(r,z)| = 2^{l-1}$ (and $|E(r,z)| = 2^{l-1}-1$). On the other hand, $E(z,z) = B$ (and $N(z,z) = \emptyset$). Hence, we get

$$\begin{aligned}
s(x) &= \frac{1}{2} + \frac{1}{2^l|B|}\sum_{r\neq h(x)}\Big((2^{l-1}-1)\cdot\mathrm{Prob}(D(f(x),r)=1) - 2^{n-1}\cdot\mathrm{Prob}(D(f(x),r)=1)\Big) \\
&\quad + \frac{1}{2^l|B|}\cdot|B|\cdot\mathrm{Prob}(D(f(x),h(x))=1) \\
&= \frac{1}{2} + \frac{1}{|B|}\cdot(\mathrm{Prob}(D(f(x),h(x))=1) - \mathrm{Prob}(D(f(x),R_n)=1))
\end{aligned}$$

Thus

$$\mathrm{Exp}(s(X_n)) = \frac{1}{2} + \frac{1}{|B|}\cdot(\mathrm{Prob}(D(f(X_n),h(X_n))=1) - \mathrm{Prob}(D(f(X_n),R_n)=1))$$

and the lemma follows. ■

2.6 * Efficient Amplification of One-way Functions

The *amplification* of weak one-way functions into strong ones, presented in Theorem 2.8, has no practical value. Recall that this amplification transforms a function f which is hard to

invert on a non-negligible fraction (i.e., $\frac{1}{p(n)}$) of the strings of length n into a function g which is hard to invert on all but a negligible fraction of the strings of length $n^2p(n)$. Specifically, it is shown that an algorithm running in time $T(n)$ which inverts g on a $\epsilon(n)$ fraction of the strings of length $n^2p(n)$ yields an algorithm running in time $\mathrm{poly}(p(n), n, \frac{1}{\epsilon(n)}) \cdot T(n)$ which inverts f on a $1 - \frac{1}{p(n)}$ fraction of the strings of length n. Hence, if f is "hard to invert in practice on a $\frac{1}{1000}$ fraction of the strings of length 100" then all we can say is that g is "hard to invert in practice on a $\frac{999}{1000}$ fraction of the strings of length 1,000,000". In contrast, an efficient amplification of one-way functions, as given below, should relate the difficulty of inverting the (weak one-way) function f on strings of length n to the difficulty of inverting the (strong one-way) function g on the strings of length $O(n)$ (rather than relating it to the to the difficulty of inverting the function g on the strings of length $\mathrm{poly}(n)$). The following definition is natural for a general discussion of amplification of one-way functions.

Definition 2.20 (quantitative one-wayness): *Let $T : \mathbb{N} \mapsto \mathbb{N}$ and $\epsilon : \mathbb{N} \mapsto \mathbf{R}$ be polynomial-time computable functions. A polynomial-time computable function $f : \{0,1\}^* \mapsto \{0,1\}^*$ is called $\epsilon(\cdot)$-one-way with respect to time $T(\cdot)$ if for every algorithm, A', with running-time bounded by $T(\cdot)$ and all sufficiently large n's*

$$\mathrm{Prob}\left(A'(f(U_n)) \notin f^{-1}f(U_n)\right) > \epsilon(n)$$

Using this terminology we review what we know already about amplification of one-way functions. A function f is weakly one-way if there exists a polynomial $p(\cdot)$ so that f is $\frac{1}{p(\cdot)}$-one-way with respect to polynomial time. A function f is strongly one-way if, for every polynomial $p(\cdot)$, the f is $(1 - \frac{1}{p(\cdot)})$-one-way with respect to polynomial time. The amplification result of Theorem 2.8 can be generalized and restated as follows. If there exist a polynomial-time computable function f which is $\frac{1}{\mathrm{poly}(\cdot)}$-one-way with respect to time $T(\cdot)$ then there exist a polynomial-time computable function g which is $(1 - \frac{1}{\mathrm{poly}(\cdot)})$-one-way with respect to time $T'(\cdot)$, where $T'(\mathrm{poly}(n)) = T(n)$ (i.e., in other words, $T'(n) = T(n^\epsilon)$ for some $\epsilon > 0$). In contrast, an efficient amplification of one-way functions, as given below, should state that the above should hold with respect to $T'(O(n)) = T(n)$ (i.e., in other words, $T'(n) = T(\epsilon \cdot n)$ for some $\epsilon > 0$). Such a result can be obtained for *regular* one-way functions. A function f is called *regular* if there exists a polynomial-time computable function $m : \mathbb{N} \mapsto \mathbb{N}$ and a polynomial $p(\cdot)$ so that, for every y in the range of f, the number of preimages (of length n) of y under f, is between $\frac{m(n)}{p(n)}$ and $m(n) \cdot p(n)$. In this book we only review the result for one-way permutations (i.e., length preserving 1-1 functions).

Theorem 2.21 (Efficient amplification of one-way permutations): *Let $p(\cdot)$ be a polynomial and $T : \mathbb{N} \mapsto \mathbb{N}$ be a polynomial-time computable function. Suppose that f is a polynomial-time computable permutation which is $\frac{1}{p(\cdot)}$-one-way with respect to time $T(\cdot)$. Then, there*

exists a polynomial-time computable permutation F so that, for every polynomial-time computable function $\epsilon:\mathbb{N}\mapsto[0,1]$, the function F is $(1-\epsilon(\cdot))$-one-way with respect to time $T'_\epsilon(\cdot)$, where $T'_\epsilon(O(n)) \stackrel{\rm def}{=} \frac{\epsilon(n)^2}{{\rm poly}(n)}\cdot T(n)$.
The constants, in the O-notation and in the poly-notation, depend on the polynomial $p(\cdot)$.

The key to the amplification of a one-way permutation f is to apply f on many different arguments. In the proof of Theorem 2.8, f is applied to unrelated arguments (which are disjoint parts of the input). This makes the proof relatively easy, but also makes the construction very inefficient. Instead, in the construction presented in the proof of the current theorem, we apply the one-way permutation f on related arguments. The first idea which comes to mind is to apply f iteratively many times, each time on the value resulting from the previous application. This will not help if easy instances for the inverting algorithm keep being mapped, by f, to themselves. We cannot just hope that this will not happen. The idea is to use randomization between successive applications. It is important that we use only a small amount of randomization,, since the "randomization" will be encoded into the argument of the constructed function. The randomization, between successive applications of f, takes the form of a random step on an expander graph. Hence a few words about these graphs and random walks on them are in place.

A graph $G=(V,E)$ is called an (n,d,c)-*expander* if it has n vertices (i.e., $|V|=n$), every vertex in V has degree d (i.e., G is d-regular), and G has the following *expansion property* (with *expansion factor* $c>0$): for every subset $S\subset V$ if $|S|\leq\frac{n}{2}$ then $|N(S)|\geq c\cdot|S|$, where $N(S)$ denotes the vertices in $V-S$ which have neighbour in S (i.e., $N(S)\stackrel{\rm def}{=}\{u\in V-S : \exists v\in S \text{ s.t. } (u,v)\in E\}$). By *explicitly constructed expanders* we mean a family of graphs $\{G_n\}_{n\in\mathbb{N}}$ so that G_n is a $(2^{2n},d,c)$ expander (d and c are the same for all graphs in the family) having a polynomial-time algorithm that on input a description of a vertex in an expander outputs its adjacency list (vertices in G_n are represented by binary strings of length $2n$). Such expender families do exist. By a *random walk* on a graph we mean the sequence of vertices visited by starting at a uniformly chosen vertex and randomly selecting at each step one of the neighbouring vertices of the current vertex, with uniform probability distribution. The expanding property implies (via a non-trivial proof) that the vertices along random walks on an expander have surprisingly strong "random properties". In particular, for every l, the probability that vertices along an $O(l)$-step long random walk hit a subset, S, is approximately the same as the probability that at least one of l independently chosen vertices hits S.

We remind the reader that we are interested in successively applying the permutation f, while interleaving randomization steps between successive applications. Hence, before applying permutation f, to the result of the previous application, we take one random step on an expender. Namely, we associate the domain of the given one-way permutation with the vertex set of the expander. Our construction alternatively applies the given one-way permutation, f, and randomly moves from the vertex just reached to one of its neighbours.

A key observation is that the composition of an expander with any permutation on its vertices yields an expander (with the same expansion properties). Combining the properties of random walks on expanders and a "reducibility" argument, the construction is showed to amplify the one-wayness of the given permutation in an efficient manner.

Construction 2.22 *Let $\{G_n\}_{n\in\mathbb{N}}$ be a family of d-regular graphs, so that G_n has vertex set $\{0,1\}^n$ and self-loops at every vertex. Consider a labeling of the edges incident to each vertex (using the labels $1,2,...,d$). Define $g_l(x)$ be the vertex reachable from vertex x by following the edge labeled l. Let $f:\{0,1\}^*\mapsto\{0,1\}^*$ be a 1-1 length preserving function. For every $k\geq 0$, $x\in\{0,1\}^n$, and $\sigma_1,\sigma_2,...,\sigma_k\in\{1,2,...,d\}$, define*

$$F(x,\sigma_1\sigma_2...\sigma_k)=\sigma_1,F(g_{\sigma_1}(f(x)),\sigma_2,...,\sigma_k)$$

(with $F(x,\lambda)=x$). For every $k:\mathbb{N}\mapsto\mathbb{N}$, define $F_{k(\cdot)}(\alpha)\stackrel{\text{def}}{=}F(x,\sigma_1,..,\sigma_t)$, where $t=k(|x|)$ and $\sigma_i\in\{1,2,...,d\}$.

Proposition 2.23 *Let $\{G_n\}$, f, $k:\mathbb{N}\mapsto\mathbb{N}$, and $F_{k(\cdot)}$ be as in Construction 2.22 (above), and suppose that $\{G_n\}_{n\in\mathbb{N}}$ is an explicitly constructed family of d-regular expander graphs, and f is polynomial-time computable. Suppose that $\alpha:\mathbb{N}\mapsto\mathbb{R}$ and $T:\mathbb{N}\mapsto\mathbb{N}$ are polynomial-time computable, and f is $\alpha(\cdot)$-one-way with respect to time $T:\mathbb{N}\to\mathbb{N}$. Then, for every polynomial-time computable $\varepsilon:\mathbb{N}\mapsto\mathbb{R}$, the function $F_{k(\cdot)}$ is polynomial-time computable as well as $(1-\varepsilon(\cdot))\beta(\cdot)$-one-way with respect to time $T':\mathbb{N}\to\mathbb{N}$, where $\beta(n)\stackrel{\text{def}}{=}(1-(1-\alpha(n))^{k(n)/2})$ and $T'(n+k(n)\cdot\log_2 d)\stackrel{\text{def}}{=}\frac{\varepsilon(n)^2\alpha(n)}{k(n)\cdot n}\cdot T(n)$.*

Theorem 2.21 follows by applying the proposition $\delta+1$ times, where δ is the degree of the polynomial $p(\cdot)$ (specified in the hypothesis that f is $\frac{1}{p(\cdot)}$-one-way). In all applications of the proposition we use $k(n)\stackrel{\text{def}}{=}3n$. In the first δ applications we use any $\varepsilon(n)<\frac{1}{7}$. The function resulting from the i^{th} application of the proposition, for $i\leq\delta$, is $\frac{1}{2n^{\delta-i}}$-one-way. In particular, after δ applications, the resulting function is $\frac{1}{2}$-one-way. (It seems that the notion of $\frac{1}{2}$-one-wayness is worthy of special attention, and deserves a name as *mostly one-way*.) In the last (i.e., $\delta+1^{\text{st}}$) application we use $\varepsilon(n)=\epsilon(n)$. The function resulting of the last (i.e., $\delta+1^{\text{st}}$) application of the proposition satisfies the statement of Theorem 2.21.

The proposition itself is proven as follows. First, we use the fact that f is a permutation to show, that the graph $G_f=(V,E_f)$, obtained from $G=(V,E)$ by letting $E_f\stackrel{\text{def}}{=}\{(u,f(v)):(u,v)\in E\}$, has the same expansion property as the graph G. Next, we use the known relation between the expansion constant of a graph and the ratio of the two largest eigenvalues of its adjacency matrix to prove that with appropriate choice of the family $\{G_n\}$ we can have this ratio bounded below by $\frac{1}{\sqrt{2}}$. Finally, we combine the following two Lemmata.

Lemma 2.24 (Random Walk Lemma): *Let G be a d-regular graph having a normalized (by factor $\frac{1}{d}$) adjacency matrix for which the ratio of the first and second eigenvalues is smaller than $\frac{1}{\sqrt{2}}$. Let $\mu \leq 1/2$ and S be a subset of measure μ of the expender's nodes. Then a random walk of length $2k$ on the expander hits S with probability at least $1-(1-\mu)^k$.*

The proof of the Random Walk Lemma regards probability distributions oven the expander vertex-set as linear combinations of the eigenvectors of the adjacency matrix. It can be shown that the largest eigenvalue is 1, and the eigenvector associated to it is the uniform distribution. Going step by step, we bound from above the probability mass assigned to random walks which do not pass through the set S. At each step, the component of the current distribution, which is in the direction of the first eigenvector, losses a factor μ of its weight (this represents the fraction of the paths which enter S in the current step). The problem is that we cannot make a similar statement with respect to the other components. Yet, using the bound on the second eigenvalue, it can be shown that in each step these components are "pushed" towards the direction of the first eigenvector. The details, being of little relevance to the topic of the book, are omitted.

Lemma 2.25 (Reducibility Lemma): *Let $\alpha, \beta : \mathbb{N} \mapsto [0,1]$, and $G_{f,n}$ be a d-regular graph on 2^n vertices satisfying the following* random path property*: for every measure $\alpha(n)$ subset, S, of $G_{f,n}$'s nodes, at least a fraction $\beta(n + k(n) \cdot \log_2 d)$ of the paths of length $k(n)$ passes through a node in S (typically $\beta(n + k(n)\log_2 d) > \alpha(n)$). Suppose that f is $(\alpha(\cdot) + \exp(\cdot))$-one-way with respect to time $T(\cdot)$. Then, for every polynomial-time computable $\varepsilon : \mathbb{N} \mapsto \mathbf{R}$, the function $F_{k(\cdot)}$, defined above, is $(1-\varepsilon(\cdot))\beta(\cdot)$-one-way with respect to time $T' : \mathbb{N} \to \mathbb{N}$, where $\beta(n + k(n)\log_2 d) \stackrel{\text{def}}{=} (1-(1-\alpha(n))^{k(n)/2})$ and $T'(n + k(n)\log_2 d) \stackrel{\text{def}}{=} \frac{\varepsilon(n)^2 \alpha(n)}{k(n)\, n} \cdot T(n)$.*

Proof Sketch: The proof is by a "reducibility argument". Assume for contradiction that $F_{k(\cdot)}$ defined as above can be inverted in time $T'(\cdot)$ with probability at least $1 - (1 - \varepsilon(m)) \cdot \beta(m)$ on inputs of length $m \stackrel{\text{def}}{=} n + k(n)\log_2 d$. Amplify A to invert $F_{k(\cdot)}$ with overwhelming probability on a $1-\beta(m)$ fraction of the inputs of length m (originally A inverts each such point with probability $> \varepsilon(m)$, as we can ignore inputs inverted with probability smaller than $\varepsilon(m)$). Note that inputs to A correspond to $k(n)$-long paths on the graph G_n. Consider the set, denoted B_n, of paths (x,p) such that A inverts $F_{k(n)}(x,p)$ with overwhelming probability.

In the sequel, we use the shorthands $k \stackrel{\text{def}}{=} k(n)$, $m \stackrel{\text{def}}{=} n + k\log_2 d$, $\varepsilon \stackrel{\text{def}}{=} \varepsilon(m)$, $\beta \stackrel{\text{def}}{=} \beta(m)$, $\alpha \stackrel{\text{def}}{=} \alpha(n)$, and $B \stackrel{\text{def}}{=} B_n$. Let P_v be the set of all k-long paths which pass through v, and B_v be the subset of B containing paths which pass through v (i.e., $B_v = B \cap P_v$). Define v as *good* if $|B_v|/|P_v| \geq \varepsilon\beta/k$ (and *bad* otherwise). Intuitively, a vertex v is called good if at least a $\varepsilon\beta/k$ fraction of the paths going through v can be inverted by A. Let $B' = B - \cup_{v \text{ bad}} B_v$; namely B' contain all "invertible" paths which pass solely through good nodes. Clearly,

Claim 2.25.1: The measure of B' in the set of all paths is greater than $1-\beta$.

Proof: Denote by $\mu(S)$ the measure of the set S in the set of all paths. Then

$$\begin{aligned}\mu(B') &= \mu(B) - \mu(\cup_{v \text{ bad}} B_v) \\ &\geq 1-(1-\epsilon)\beta - \sum_{v \text{ bad}} \mu(B_v) \\ &> 1-\beta+\epsilon\beta - \sum_{v}(\epsilon\beta/k)\mu(P_v) \\ &> 1-\beta \quad \square\end{aligned}$$

Using the random path property, we have

Claim 2.25.2: The measure of good nodes is at least $1-\alpha$.

Proof: Otherwise, let S be the set of bad nodes. If S has measure α then, by the random path property, it follows the fraction of path which pass through vertices of S is at least β. Hence, B', which cannot contain such paths can contain only a $1-\beta$ fraction of all paths in contradiction to Claim 2.25.1. $\square$

The following algorithm for inverting f, is quite natural. The algorithm uses as subroutine an algorithm, denoted A, for inverting $F_{k(\cdot)}$. Inverting f on y is done by placing y on a random point along a randomly selected path $\overline{p}$, taking a walk from y according to the suffix of $\overline{p}$, and asking A for the preimage of the resulting pair under F_k.

Algorithm for inverting f:
On input y, `repeat` $\frac{kn}{\epsilon\beta}$ `times`:

1. Select randomly $i \in \{1,2,...,k\}$, and $\sigma_1, \sigma_2, ..., \sigma_k \in \{1,2,...,d\}$;
2. Compute $y' = F(g_{\sigma_i}(y), \sigma_{i+1}...\sigma_k)$;
3. Invoke A to get $x' \leftarrow A(\sigma_1\sigma_2, ..., \sigma_k, y')$;
4. Compute $x = F(x', \sigma_1...\sigma_{i-1})$;
5. If $f(x) = y$ then halt and output x.

Analysis of the inverting algorithm (for a good x):

Since x is good, a random path going through it (selected above) corresponds to an "invertible path" with probability at least $\epsilon\beta/k$. If such a path is selected then we obtain the inverse of $f(x)$ with overwhelming probability. The algorithm for inverting f repeats the process sufficiently many times to guarantee overwhelming probability of selecting an "invertible path".

By Claim 2.25.2, the good x's constitute a $1-\alpha$ fraction of all n-bit strings. Hence, the existence of an algorithm inverting $F_{k(\cdot)}$, in time $T'(\cdot)$ with probability at least $1-(1-\varepsilon(\cdot))\beta(\cdot)$, implies the existence of an algorithm inverting f, in time $T(\cdot)$ with probability at least $1-\alpha(\cdot)-\exp(\cdot)$. This constitutes a contradiction to the hypothesis of the lemma, and hence the lemma follows. ■

2.7 Miscellaneous

2.7.1 Historical Notes

The notion of a one-way function originates from the paper of Diffie and Hellman [DH76]. Weak one-way functions were introduced by Yao [Y82]. The RSA function was introduced by Rivest, Shamir and Adleman [RSA78], whereas squaring modulo a composite was introduced and studied by Rabin [R79]. The suggestion for basing one-way functions on the believed intractability of decoding random linear codes is taken from [BMT78,GKL88], and the suggestion to base one-way functions on the subset sum problem is taken from [IN89].

The equivalence of existence of weak and strong one-way functions is implicit in Yao's work [Y82]. The existence of universal one-way functions is stated in Levin's work [L85]. The efficient amplification of one-way functions, presented in Section 2.6, is taken from Goldreich el. al. [GILVZ], which in turn uses ideas originating in [AKS].

> **Author's Note:** *GILVZ = Goldreich, Impagliazzo, Levin, Venkatesan and Zuckerman (FOCS90); AKS = Ajtai, Komolos and Szemeredi (STOC87).*

The concept of hard-core predicates originates from the work of Blum and Micali [BM82]. That work also proves that a particular predicate constitutes a hard-core for the "DLP function" (i.e., exponentiation in a finite field), provided that this function is one-way. Consequently, Yao proved that the existence of one-way functions implies the existence of hard-core predicates [Y82]. However, Yao's construction, which is analogous to the contraction used for the proof of Theorem 2.8, is of little practical value. The fact that the inner-product mod 2 is a hard-core for any one-way function (of the form $g(x,r)=(f(x),r)$) was proven by Goldreich and Levin [GL89]. The proof presented in this book, which follows ideas originating in [ACGS84], is due to Charles Rackoff.

Hard-core predicates and functions for specific collections of permutations were suggested in [BM82,LW,K88,ACGS84,VV84]. Specifically, Kalisky [K88], extending ideas of [BM82,LW], proves that the intractability of various discrete logarithm problems yields hard-core functions for the related exponentiation permutations. Alexi el. al. [ACGS84], building on work by Ben-Or et. al. [BCS83], prove that the intractability of factoring yields hard-core functions for permutations induced by squaring modulo a composite number.

2.7.2 Suggestion for Further Reading

Our exposition of the RSA and Rabin functions is quite sparse in details. In particular, the computational problems of generating uniformly distributed "certified primes" and of "primality checking" deserve much more attention. A probabilistic polynomial-time algorithm for generating uniformly distributed primes together with corresponding certificates of primality has been presented by Bach [BachPhd]. The certificate produced, by this algorithm, for a prime P consists of the prime factorization of $P-1$, together with certificates for primality of these factors. This recursive form of certificates for primality originates in von-Pratt's proof that the set of primes is in $\mathcal{NP}$ (cf. [vP]). However, the above procedure is not very practical. Instead, when using the RSA (or Rabin) function in practice, one is likely to prefer an algorithm that generates integers at random and checks them for primality using fast primality checkers such as the algorithms presented in [SSprime,Rprime]. One should note, however, that these algorithms do not produce certificates for primality, and that with some (small) probability may assert that a composite number is a prime. Probabilistic polynomial-time algorithms (yet not practical ones) that, given a prime, produce a certificate for primality, are presented in [GKprime,AHprime]

> **Author's Note:** *SSprime = Solovay and Strassen, Rprime = Rabin, GKprime = Goldwasser and Kilian, AHprime = Adleman and Haung.*

The subset sum problem is known to be easy in two special cases. One case is the case in which the input sequence is constructed based on a simple "hidden sequence". For example, Merkle and Hellman [MH78], suggested to construct an instance of the subset-sum problem based on a "hidden super increasing sequence" as follows. Let $s_1, ..., s_n, M \stackrel{\text{def}}{=} s_{n+1}$ be a sequence satisfying, $s_i > \sum_{j=1}^{i-1} s_j$, for every i, and let w be relatively prime to M. Such a sequence is called *super increasing*. The instance consists of $(x_1, ..., x_n)$ and $\sum_{i \in I} x_i$, for $I \subseteq \{1, ..., n\}$, where $x_i \stackrel{\text{def}}{=} w \cdot s_i \bmod M$. It can be shown that knowledge of both w and M allows easy solution of the subset sum problem for the above instance. The hope was that, when w and M are not given, solving the subset-sum problem is hard even for instances generated based on a super increasing sequence (and this would lead to a trapdoor one-way function). However, the hope did not materialize. Shamir presented an efficient algorithm for solving the subset-sum problem for instances with a hidden super increasing sequence [S82]. Another case for which the subset sum problem is known to be easy is the case of *low density* instances. In these instances the length of the elements in binary representation is considerably larger than the number of elements (i.e. $|x_1| = \cdots = |x_n| = (1+\epsilon)n$ for some constant $\epsilon > 0$). For further details consult the original work of Lagarias and Odlyzko [LO85] and the later survey of Brickell and Odlyzko [BO88].

For further details on hard-core functions for the RSA and Rabin functions the reader is directed to Alexi el. al. [ACGS84]. For further details on hard-core functions for the "DLP function" the reader is directed to Kalisky's work [K88].

The theory of average-case complexity, initiated by Levin [L84], is somewhat related to the notion of one-way functions. For a survey of this theory we refer the reader to [BCGL]. Loosely speaking, the difference is that in our context it is required that the (efficient) "generator" of hard (on-the-average) instances can easily solve them himself, whereas in Levin's work the instances are hard (on-the-average) to solve even for the "generator". However, the notion of average-case reducibility introduced by Levin is relevant also in our context.

Author's Note: *BCGL = Ben-David, Chor, Goldreich and Luby (JCSS, April 1992).*

Readers interested in further details about the best algorithms known for the factoring problem are directed to Pomerance's survey [P82]. Further details on the best algorithms known for the discrete logarithm problem (DLP) can be found in Odlyzko's survey [O84]. In addition, the reader is referred to Bach and Shalit's book on computational number theory [BS92book]. Further details about expander graphs, and random walks on them, can be found in the book of Alon and Spencer [AS91book].

Author's Note: *Updated versions of the surveys by Pomerance and Odlyzko do exist.*

2.7.3 Open Problems

The efficient amplification of one-way functions, originating in [GILVZ], is only known to work for special types of functions (e.g., regular ones). We believe that presenting (and proving) an efficient amplification of arbitrary one-way functions is a very important open problem. It may also be instrumental for more efficient constructions of pseudorandom generators based on arbitrary one-way functions (see Section 3.5).

An open problem of more practical importance is to try to present hard-core functions with larger range for the RSA and Rabin functions. Specifically, assuming that squaring mod N is one-way, is the function which returns the first half of x a hard-core of squaring mod N? Some support to a positive answer is provided by the work of Shamir and Shrift [SS90]. A positive answer would allow to construct extremely efficient pseudorandom generators and public-key encryption schemes based on the conjectured intractability of the factoring problem.

2.7.4 Exercises

Exercise 1: *Closing the gap between the motivating discussion and the definition of one-way functions*: We say that a function $h:\{0,1\}^* \mapsto \{0,1\}^*$ is *hard on the average but*

easy with auxiliary input if there exists a probabilistic polynomial-time algorithm, G, such that

1. There exists a polynomial-time algorithm, A, such that $A(x,y) = h(x)$ for every (x,y) in the range of G (i.e., for every (x,y) so that (x,y) is a possible output of $G(1^n)$ for some input 1^n).
2. for every probabilistic polynomial-time algorithm, A', every polynomial $p(\cdot)$, and all sufficiently large n's

$$\mathrm{Prob}(A'(X_n) = h(X_n)) < \frac{1}{p(n)}$$

where $(X_n, Y_n) \stackrel{\mathrm{def}}{=} G(1^n)$ is a random variable assigned the output of G.

Prove that if there exist "hard on the average but easy with auxiliary input" functions then one-way functions exist.

Exercise 2: *One-way functions and the $\mathcal{P}$ vs. $\mathcal{NP}$ question (part 1):* Prove that the existence of one-way functions implies $\mathcal{P} \neq \mathcal{NP}$.
(Guidelines: for every function f define $L_f \in \mathcal{NP}$ so that if $L_f \in \mathcal{P}$ then there exists a polynomial-time algorithm for inverting f.)

Exercise 3: *One-way functions and the $\mathcal{P}$ vs. $\mathcal{NP}$ question (part 2):* Assuming that $\mathcal{P} \neq \mathcal{NP}$, construct a function f so that the following three claims hold:

1. f is polynomial-time computable;
2. there is no polynomial-time algorithm that always inverts f (i.e., successfully inverts f on every y in the range of f); and
3. f is not (even weakly) one-way. Furthermore, there exists a polynomial-time algorithm which inverts f with exponentially small failure probability, where the probability space is (again) of all possible choices of input (i.e., $f(x)$) and internal coin tosses for the algorithm.

(Guidelines: consider the function f_{sat} defined so that $f_{\mathrm{sat}}(\phi,\tau) = (\phi,1)$ if τ is a satisfying assignment to propositional formulae ϕ, and $f_{\mathrm{sat}}(\phi,\tau) = (\phi,0)$ otherwise. Modify this function so that it is easy to invert on most instances, yet inverting f_{sat} is reducible to inverting its modification.)

Exercise 4: Let f be a strongly one-way function. Prove that for every probabilistic polynomial-time algorithm A, and for every polynomial $p(\cdot)$ the set

$$B_{A,p} \stackrel{\mathrm{def}}{=} \{x : \mathrm{Prob}(A(f(x)) \in f^{-1}f(x)) \geq \frac{1}{p(|x|)}\}$$

has negligible density in the set of all strings (i.e., for every polynomial $q(\cdot)$ and all sufficiently large n it holds that $\frac{|B \cap \{0,1\}^n}{2^n} < \frac{1}{p(n)}$).

Exercise 5: *Another definition of non-uniformly one-way functions*: Consider the definition resulting from Definition 2.6 by allowing the circuits to be probabilistic (i.e., have an auxiliary input which is uniformly selected). Prove that the resulting new definition is equivalent to the original one.

Exercise 6: Let f_{mult} be as defined in Section 2.2. Assume that every integer factoring algorithm has, on input N, running time $L(P) \stackrel{\text{def}}{=} 2^{\Theta(\sqrt{\log P \log\log P})}$, where P is the second biggest prime factor of N. Prove that f_{mult} is strongly one-way.
(Guideline: using results on density of smooth numbers, show that the density, of integers N with second biggest prime smaller than $L(N)$, is smaller that $\frac{1}{L(N)}$.)

Exercise 7: Define $f_{\text{add}} : \{0,1\}^* \mapsto \{0,1\}^*$ so that $f_{\text{add}}(xy) = \text{prime}(x) + \text{prime}(y)$, where $|x| = |y|$ and $\text{prime}(z)$ is the smallest prime which is larger than z. Prove that f_{add} is not a one-way function.
(Guideline: don't try to capitalize on the possibility that $\text{prime}(N)$ is too large, e.g., larger than $N + \text{poly}(\log N)$. It is unlikely that such a result, in number theory, can be proven. Furthermore, it is generally believed that there exists a constant c such that, for all integer $N \geq 2$, it holds that $\text{prime}(N) < N + \log_2^c N$.) Hence, it is likely that f_{add} is polynomial-time computable.)

Exercise 8: Prove that *one-way functions cannot have a polynomial-size range*. Namely, prove that if f is (even weakly) one-way then for every polynomial $p(\cdot)$ and all sufficiently large n's it holds $|\{f(x) : x \in \{0,1\}^n\}| > p(n)$.

Exercise 9: Prove that *one-way functions cannot have polynomially bounded cycles*. Namely, for every function f define $\text{cyc}_f(x)$ to be the smallest positive integer i such that applying f for i times on x yields x. Prove that if f is (even weakly) one-way then for every polynomial $p(\cdot)$ and all sufficiently large n's it holds $\text{Exp}(\text{cyc}_f(U_n)) > p(n)$, where U_n is a random variable uniformly distributed over $\{0,1\}^n$.

Exercise 10: *on the improbability of strengening Theorem 2.8 (part 1)*: Suppose that the definition of weak one-way function is further weakened so that it is required that every algorithm fails to inverts the function with negligible probability. Demonstrate the difficulty of extending the proof of Theorem 2.8 to this case.
(Hint: suppose that there exists an algorithm that if run with time bound $t(n)$ inverts the function with probability $1/t(n)$.)

Exercise 11: *on the improbability of strengening Theorem 2.8 (part 2)* (due to S. Rudich): Suppose that the definition of a strong one-way function is further strengthen so that it is required that every algorithm fails to inverts the function with some *specified* negligible probability (e.g., $2^{-\sqrt{n}}$). Demonstrate the difficulty of extending the proof of Theorem 2.8 to this case.
(Guideline: suppose that that we construct the strong one-way function g as in the

original proof. Note that you *can* prove that any algorithm that works separately on each block of the function g, can invert it only with exponentially low probability. However, there may be an inverting algorithm, A, that inverts the function g with probability ϵ. Show that any inverting algorithm for the weakly one-way function f that uses algorithm A as a black-box "must" invoke it at least $\frac{1}{\epsilon}$ times.)

Exercise 12: *collections of one-way functions and one-way functions*: Represent a collection of one-way functions, (I, D, F), as a single one-way function. Given a one-way function f, represent it as a collection of one-way functions.
(Remark: the second direction is quite trivial.)

Exercise 13: *a convention for collections of one-way functions*: Show that without loss of generality, algorithms I and D of a collection (of one-way functions) can be modified so that each of them uses a number of coins which exactly equals the input length.
(Guideline: "apply padding" first on 1^n, next on the coin tosses and output of I, and finally to the coin tosses of D.)

Exercise 14: *justification for a convention concerning one-way collections*: Show that giving the index of the function to the inverting algorithm is essential for a meaningful definition of a collection of one-way functions.
(Guideline: consider a collection $\{f_i : \{0,1\}^{|i|} \mapsto \{0,1\}^{|i|}\}$ where $f_i(x) = x \oplus i$.)

Exercise 15: *Rabin's collection and factoring*: Show that the Rabin collection is one-way if and only if factoring integers which are the product of two primes of equal binary expsansion is intractable in a strong sense (i.e., every efficient algorithm succeeds with negligible probability).
(Guideline: For one direction use the Chinese Reminder Theorem and an efficient algorithm for extracting square roots modulo a prime. For the other direction observe that an algorithm for extracting square roots modulo a composite N can be use to get two integers x and y such that $x^2 \equiv y^2 \bmod N$ and yet $x \not\equiv \pm y \bmod N$. Also, note that such a pair, (x, y), yields a split of N (i.e., two integers $a, b \neq 1$ such that $N = a \cdot b$).)

Exercise 16: *clawfree collections imply one-way functions*: Let (I, D, F) be a clawfree collection of functions (see Subsection 2.4.5). Prove that, for every $\sigma\{0,1\}$, the triplet (I, D, F_σ), where $F_\sigma(i, x) \stackrel{\text{def}}{=} F(\sigma, i, x)$, is a collection of strong one-way functions. Repeat the exercise when replacing the word 'functions' by 'permutations'. (I, D, F) be a clawfree collection of functions

Exercise 17: *more on the inadequacy of graph isomorphism as a basis for one-way functions*: Consider another suggestion to base one-way functions on the conjectured difficulty of the Graph Isomorphism problem. This time we present a collection of functions, defined by the algorithmic triplet $(I_{\mathrm{GI}}, D_{\mathrm{GI}}, F_{\mathrm{GI}})$. On input 1^n, algorithm

$I_{\rm GI}$ selects uniformly a $d(n)$-regular graph on n vertices (i.e., each of the n vertices in the graph has degree $d(n)$). On input a graph on n vertices, algorithm $D_{\rm GI}$ randomly selects a permutation in the symmetric group of n elements (i.e., the set of permutations of n elements). On input a (n-vertex) graph G and a (n-element) permutation π, algorithm $F_{\rm GI}$ returns $f_G(\pi) \stackrel{\rm def}{=} \pi G$.

1. Present a polynomial-time implementation of $I_{\rm GI}$.
2. In light of the known algorithms for the Graph Isomorphism problem, which values of $d(n)$ should be definitely avoided?
3. Using a known algorithm, prove that the above collection does not have a one-way property, no matter which function $d(\cdot)$ one uses.

(A search into the relevant literature is indeed required for items (2) and (3).)

Exercise 18: Assuming the existence of one-way functions, prove that there exist a one-way function f so that *no* single bit of the preimage constitutes a hard-core predicate. (Guideline: given a one-way function f construct a function g so that $g(x,I,J) \stackrel{\rm def}{=} (f(x_{I\cap J}), x_{\overline{I}\cup\overline{J}}, I, J)$, where $I, J \subseteq \{1,2,...|x|\}$, and x_S denotes the string resulting by taking only the bits of x with positions in the set S (i.e., $x_{i_1,...,i_s} \stackrel{\rm def}{=} x_{i_1}\cdots x_{i_s}$, where $x = x_1\cdots x_{|x|}$).)

Exercise 19: *hard-core predicate for a 1-1 function implies that the function is one-way*: Let f be a 1-1 function (you may assume for simplicity that it is length preserving) and let b be a hard-core for f.

1. Prove that if f is polynomial-time computable then it is strongly one-way.
2. Prove that (regardless of whether f is polynomial-time computable or not) f must be weakly one-way. Furthermore, for every $\delta > \frac{1}{2}$, the function f cannot be inverted on a δ fraction of the instances.

Exercise 20: In continuation to the proof of Theorem 2.15, we present guidelines for a more efficient inverting algorithm. In the sequel it will be more convenient to use arithmetic of reals instead of that of Boolean. Hence, we denote $b'(x,r) = (-1)^{b(r,x)}$ and $G'(y,r) = (-1)^{G(y,r)}$.

1. Prove that for every x it holds that $\mathrm{Exp}(b'(x,r)\cdot G'(f(x),r+e^i)) = s'(x)\cdot(-1)^{x_i}$, where $s'(x) \stackrel{\rm def}{=} 2\cdot(s(x)-\frac{1}{2})$.
2. Let v be an l-dimensional Boolean vector, and let R be a uniformly chosen l-by-n Boolean matrix. Prove that for every $v \neq u \in \{0,1\}^l$ it holds that vR and uR are pairwise independent and uniformly distributed in $\{0,1\}^n$.
3. Prove that $b'(x,vR) = b'(xR^T, v)$, for every $x \in \{0,1\}^n$ and $v \in \{0,1\}^l$.

4. Prove that, with probability at least $\frac{1}{2}$, there exists $\sigma \in \{0,1\}^l$ so that for every $1 \leq i \leq n$ the sign of $\sum_{v \in \{0,1\}^l} b'(\sigma, v) G'(f(x), vR + e^i))$ equals the sign of $(-1)^{x_i}$. (Hint: $\sigma \stackrel{\text{def}}{=} xR^T$.)

5. Let B be an 2^l-by-2^l matrix with the (σ, v)-entry being $b'(\sigma, v)$, and let $\overline{g}^i$ be an 2^l-dimensional vector with the v^{th} entry equal $G'(f(x), vR + e^i)$. The inverting algorithm computes $\overline{z}_i \leftarrow Bg^i$, for all i's, and forms a matrix Z in which the columns are the $\overline{z}_i$'s. The output is a row that when applying f to it yields $f(x)$. Evaluate the success probability of the algorithm. Using the special structure of matrix B, show that the product Bg^i can be computed in time $l \cdot 2^l$.
Hint: B is the Sylvester matrix, which can be written recursively as

$$S_k = \begin{pmatrix} S_{k-1} & S_{k-1} \\ S_{k-1} & \overline{S_{k-1}} \end{pmatrix}$$

where $S_0 = +1$ and $\overline{M}$ means flipping the $+1$ entries of M to -1 and vice versa.

Chapter 3

Pseudorandom Generators

In this chapter we discuss pseudorandom generators. Loosely speaking, these are efficient deterministic programs which expand short randomly selected seeds into much longer "pseudorandom" bit sequences. Pseudorandom sequences are defined as computationally indistinguishable from truly random sequences by efficient algorithms. Hence, the notion of computational indistinguishability (i.e., indistinguishability by efficient procedures) plays a pivotal role in our discussion of pseudorandomness. Furthermore, the notion of computational indistinguishability, plays a key role also in subsequent chapters, and in particular in the discussion of secure encryption, zero-knowledge proofs, and cryptographic protocols.

In addition to definitions of pseudorandom distributions, pseudorandom generators, and pseudorandom functions, the current chapter contains constructions of pseudorandom generators (and pseudorandom functions) based on various types of one-way functions. In particular, very simple and efficient pseudorandom generators are constructed based on the existence of one-way permutations.

3.1 Motivating Discussion

The nature of randomness has attracted the attention of many people and in particular of scientists in various fields. We believe that the notion of computation, and in particular of efficient computation, provides a good basis for understanding the nature of randomness.

3.1.1 Computational Approaches to Randomness

One computational approach to randomness has been initiated by Solomonov and Kolmogorov in the early 1960's (and rediscovered by Chaitin in the early 1970's). This approach is "ontological" in nature. Loosely speaking, a string, s, is considered *Kolmogorov-random*

if its length (i.e., $|s|$) equals the length of the shortest program producing s. This shortest program may be considered the "simplest" "explanation" to the phenomenon described by the string s. Hence, the string, s, is considered Kolmogorov-random if it does not posses a simple explanation (i.e., an explanation which is substantially shorter than $|s|$). We stress that one cannot determine whether a given string is Kolmogorov-random or not (and more generally Kolmogorov-complexity is a function that cannot be computed). Furthermore, this approach seems to have no application to the issue of "pseudorandom generators".

An alternative computational approach to randomness is presented in the rest of this chapter. In contrast to the approach of Kolmogorov, the new approach is behavioristic in nature. Instead of considering the "explanation" to a phenomenon, we consider the phenomenon's effect on the environment. Loosely speaking, a string is considered *pseudorandom* if no efficient observer can distinguish it from a uniformly chosen string of the same length. The underlying postulate is that objects that cannot be told apart by efficient procedures are considered equivalent, although they may be very different in nature (e.g., have fundamentally different (Kolmogorov) complexity). Furthermore, the new approach naturally leads to the concept of a pseudorandom generator, which is a fundamental concept with lots of practical applications (and in particular to the area of cryptography).

3.1.2 A Rigorous Approach to Pseudorandom Generators

The approach to pseudorandom generators, presented in this book, stands in contrast to the heuristic approach which is still common in discussions concerning "pseudorandom generators" which are being used in real computers. The heuristic approach consider "pseudorandom generators" as programs which produce bit sequences "passing" *several* specific statistical tests. The choice of statistical tests, to which these programs are subjected, is quite arbitrary and lacks a systematic foundation. Furthermore, it is possible to construct efficient statistical tests which foil the "pseudorandom generators" commonly used in practice (and in particular distinguish their output from a uniformly chosen string of equal length). Consequently, before using a "pseudorandom generator", in a new application (which requires "random" sequences), extensive tests have to be conducted in order to detect whether the behaviour of the application when using the "pseudorandom generator" preserves its behaviour when using a "true source of randomness". Any modification of the application requires new comparison of the "pseudorandom generator" against the "random source", since the non-randomness of the "pseudorandom generator" may badly effect the modified application (although it did not effect the original application). Furthermore, using such a "pseudorandom generator" for "cryptographic purposes" is highly risky, since the adversary may try to exploit the known weaknesses of the "pseudorandom generator".

In contrast the concept of pseudorandom generators, presented below, is a robust one. By definition these pseudorandom generators produce sequences which look random to any efficient observer. It follows that the output of a pseudorandom generator may be used

instead of "random sequences" in any efficient application requiring such (i.e., "random") sequences.

3.2 Computational Indistinguishability

The concept of efficient computation leads naturally to a new kind of equivalence between objects. *Objects are considered to be computationally equivalent if they cannot be told apart by any efficient procedure.* Considering indistinguishable objects as equivalent is one of the basic paradigms of both science and real-life situations. Hence, we believe that the notion of computational indistinguishability is fundamental.

Formulating the notion of computational indistinguishability is done, as standard in computational complexity, by considering objects as infinite sequences of strings. Hence, the sequences, $\{x_n\}_{n\in N}$ and $\{y_n\}_{n\in N}$, are said to be computational indistinguishable if no efficient procedure can tell them apart. In other words, no efficient algorithm, D, can accept infinitely many x_n's while rejecting their y-counterparts (i.e., for every efficient algorithm D and all sufficiently large n's it holds that D accepts x_n iff D accepts y_n). Objects which are computationally indistinguishable in the above sense may be considered equivalent as far as any practical purpose is concerned (since practical purposes are captured by efficient algorithms and those can not distinguish these objects).

The above discussion is naturally extended to the probabilistic setting. Furthermore, as we shall see, this extension yields very useful consequences. Loosely speaking, two distributions are called computationally indistinguishable if no efficient algorithm can tell them apart. Given an efficient algorithm, D, we consider the probability that D accepts (e.g., outputs 1 on input) a string taken from the first distribution. Likewise, we consider the probability that D accepts a string taken from the second distribution. If these two probabilities are close, we say that D does not distinguish the two distributions. Again, the formulation of this discussion is with respect to two infinite sequences of distributions (rather than with respect to two fixed distributions). Such sequences are called probability ensembles.

3.2.1 Definition

Definition 3.1 (ensembles): *Let I be a countable index set. An* ensemble indexed by I *is a sequence of random variables indexed by I. Namely, $X = \{X_i\}_{i\in I}$, where the X_i's are random variables, is an ensemble indexed by I.*

We will use either $\mathbb{N}$ or a subset of $\{0,1\}^*$ as the index set. Typically, in our applications, an ensemble of the form $X = \{X_n\}_{n\in\mathbb{N}}$ has each X_n ranging over strings of length n, whereas an ensemble of the form $X = \{X_w\}_{w\in\{0,1\}^*}$ will have each X_w ranging over strings

of length $|w|$. In the rest of this chapter, we will deal with ensembles indexed by $\mathbb{N}$, whereas in other chapters (e.g., in the definition of secure encryption and zero-knowledge) we will deal with ensembles indexed by strings. To avoid confusion, we present variants of the definition of computational indistinguishability for each of these two cases. The two formulations can be unified if one associates the natural numbers with their unary representation (i.e., associate $\mathbb{N}$ and $\{1^n : n \in \mathbb{N}\}$).

Definition 3.2 (polynomial-time indistinguishability):

1. variant for ensembles indexed by $\mathbb{N}$: *Two ensembles,* $X \stackrel{\text{def}}{=} \{X_n\}_{n\in\mathbb{N}}$ *and* $Y \stackrel{\text{def}}{=} \{Y_n\}_{n\in\mathbb{N}}$, *are* indistinguishable in polynomial-time *if for every probabilistic polynomial-time algorithm,* D, *every polynomial* $p(\cdot)$, *and all sufficiently large* n*'s*

$$|\text{Prob}\,(D(X_n, 1^n) = 1) - \text{Prob}\,(D(Y_n, 1^n) = 1)\,| < \frac{1}{p(n)}$$

2. variant for ensembles indexed by a set of strings S: *Two ensembles,* $X \stackrel{\text{def}}{=} \{X_w\}_{w\in S}$ *and* $Y \stackrel{\text{def}}{=} \{Y_w\}_{w\in S}$, *are* indistinguishable in polynomial-time *if for every probabilistic polynomial-time algorithm,* D, *every polynomial* $p(\cdot)$, *and all sufficiently long* w*'s*

$$|\text{Prob}\,(D(X_w, w) = 1) - \text{Prob}\,(D(Y_w, w) = 1)\,| < \frac{1}{p(|w|)}$$

The probabilities in the above definition are taken over the corresponding random variables X_i (or Y_i) and the internal coin tosses of algorithm D (which is allowed to be a probabilistic algorithm). The second variant of the above definition will play a key role in subsequent chapters, and further discussion of it is postponed to these places. In the rest of this chapter we refer only to the first variant of the above definition. The string 1^n is given as auxiliary input to algorithm D in order to make the first variant consistent with the second one, and in order to make it more intuitive. However, *in typical cases*, where the length of X_n (resp. Y_n) and n are polynomialy related (i.e., $|X_n| < \text{poly}(n)$ and $n < \text{poly}(|X_n|)$) and can be computed one from the other in poly(n)-time, *giving* 1^n *as auxiliary input is redundant.*

The following mental experiment may be instructive. For each $\alpha \in \{0,1\}^*$, consider the probability, hereafter denoted $d(\alpha)$, that algorithm D outputs 1 on input α. Consider the expectation of d taken over each of the two ensembles. Namely, let $d_1(n) = \text{Exp}(d(X_n))$ and $d_2(n) = \text{Exp}(d(Y_n))$. Then, X and Y are said to be indistinguishable by D if the difference (function) $\Delta(n) \stackrel{\text{def}}{=} |d_1(n) - d_2(n)|$ is negligible in n. A few examples may help to further clarify the definition.

Consider an algorithm, D_1, which obliviously of the input, flips a 0-1 coin and outputs its outcome. Clearly, on every input, algorithm D_1 outputs 1 with probability exactly one

half, and hence does not distinguish any pair of ensembles. Next, consider an algorithm, D_2, which outputs 1 if and only if the input string contains more zeros than ones. Since D_2 can be implemented in polynomial-time, it follows that if X and Y are polynomial-time indistinguishable then the difference $|\mathrm{Prob}(\omega(X_n) < \frac{n}{2}) - \mathrm{Prob}(\omega(Y_n) < \frac{n}{2})|$ is negligible (in n), where $\omega(\alpha)$ denotes the number of 1's in the string α. Similarly, polynomial-time indistinguishable ensembles must exhibit the same "profile" (up to negligible error) with respect to any "string statistics" which can be computed in polynomial-time. However, it is not required that polynomial-time indistinguishable ensembles have similar "profiles" with respect to quantities which cannot be computed in polynomial-time (e.g., Kolmogorov Complexity or the function presented right after Proposition 3.3).

3.2.2 Relation to Statistical Closeness

Computational indistinguishability is a refinement of a traditional notion from probability theory. We call two ensembles $X \stackrel{\mathrm{def}}{=} \{X_n\}_{n\in\mathbb{N}}$ and $Y \stackrel{\mathrm{def}}{=} \{Y_n\}_{n\in\mathbb{N}}$, *statistically close* if their statistical difference is negligible, where the *statistical difference* (also known as *variation distance*) of X and Y is defined as the function

$$\Delta(n) \stackrel{\mathrm{def}}{=} \sum_{\alpha} |\mathrm{Prob}(X_n = \alpha) - \mathrm{Prob}(Y_n = \alpha)|$$

Clearly, if the ensembles X and Y are statistically close then they are also polynomial-time indistinguishable (see Exercise 5). The converse, however, is not true. In particular

Proposition 3.3 *There exist an ensemble $X = \{X_n\}_{n\in\mathbb{N}}$ so that X is not statistically close to the uniform ensemble, $U \stackrel{\mathrm{def}}{=} \{U_n\}_{n\in\mathbb{N}}$, yet X and U are polynomial-time indistinguishable. Furthermore, X_n assigns all its probability mass to at most $2^{n/2}$ strings (of length n).*

Recall that U_n is uniformly distributed over strings of length n. Although X and U are polynomial-time indistinguishable, one can define a function $f : \{0,1\}^* \mapsto \{0,1\}$ so that f has average 1 over X while having average almost 0 over U (e.g., $f(x) = 1$ if and only if x is in the range of X). Hence, X and U have different "profile" with respect to the function f, yet f is (necessarily) impossible to compute in polynomial-time.

Proof: We claim that, for all sufficiently large n, there exist a random variable X_n, distributed over some set of at most $2^{n/2}$ strings (each of length n), so that for every circuit, C_n, of size (i.e., number of gates) $2^{n/8}$ it holds that

$$|\mathrm{Prob}(C_n(U_n) = 1) - \mathrm{Prob}(C_n(X_n) = 1)| < 2^{-n/8}$$

The proposition follows from this claim, since polynomial-time distinguishers (even probabilistic ones - see Exercise 6) yield polynomial-size circuits with at least as big a distinguishing gap.

The claim is proven using a probabilistic argument (i.e., a counting argument). Let C_n be some fixed circuit with n inputs, and let $p_n \stackrel{\text{def}}{=} \text{Prob}(C_n(U_n) = 1)$. We select, independently and uniformly $2^{n/2}$ strings, denoted $s_1, ..., s_{2^{n/2}}$, in $\{0,1\}^n$. Define random variables ζ_i's so that $\zeta_i = C_n(s_i)$ (these random variables depend on the random choices of the corresponding s_i's). Using Chernoff Bound, we get that

$$\text{Prob}\left(|p_n - \frac{1}{2^{n/2}} \cdot \sum_{i=1}^{2^{n/2}} \zeta_i| \geq 2^{-n/8}\right) \leq 2e^{-2 \cdot 2^{n/2} \cdot 2^{-n/4}} < 2^{-2^{n/4}}$$

Since there are at most $2^{2^{n/4}}$ different circuits of size (number of gates) $2^{n/8}$, it follows that there exists a sequence of $s_1, ..., s_{2^{n/2}} \in \{0,1\}^n$, so that for every circuit C_n of size $2^{n/8}$ it holds that

$$|\text{Prob}(C_n(U_n) = 1) - \frac{1}{2^{n/2}} \sum_{i=1}^{2^{n/2}} C_n(s_i)| < 2^{-n/8}$$

Letting X_n equal s_i with probability $2^{-n/2}$, for every $1 \leq i \leq 2^{n/2}$, the claim follows. ■

3.2.3 Indistinguishability by Repeated Experiments

By Definition 3.2, two ensembles are considered computationally indistinguishable if no efficient procedure can tell them apart based on a single sample. We shall now show that "efficiently constructible" computational indistinguishable ensembles cannot be (efficiently) distinguished even by examining several samples. We start by presenting definitions of "indistinguishability by sampling" and "efficiently constructible ensembles".

Definition 3.4 (indistinguishability by sampling): *Two ensembles, $X \stackrel{\text{def}}{=} \{X_n\}_{n \in \mathbb{N}}$ and $Y \stackrel{\text{def}}{=} \{Y_n\}_{n \in \mathbb{N}}$, are* indistinguishable by polynomial-time sampling *if for every probabilistic polynomial-time algorithm, D, every two polynomials $m(\cdot)$ and $p(\cdot)$, and all sufficiently large n's*

$$|\text{Prob}\left(D(X_n^{(1)}, ..., X_n^{(m(n))}) = 1\right) - \text{Prob}\left(D(Y_n^{(1)}, ..., Y_n^{(m(n))}) = 1\right)| < \frac{1}{p(n)}$$

where $X_n^{(1)}$ through $X_n^{(m)}$ and $Y_n^{(1)}$ through $Y_n^{(m)}$, are independent random variables with each $X_n^{(i)}$ identical to X_n and each $Y_n^{(i)}$ identical to Y_n.

Definition 3.5 (efficiently constructible ensembles): *An ensemble, $X \stackrel{\text{def}}{=} \{X_n\}_{n\in\mathbb{N}}$, is said to be* polynomial-time constructible *if there exists a probabilistic polynomial time algorithm S so that for every n, the random variables $S(1^n)$ and X_n are identically distributed.*

Theorem 3.6 *Let $X \stackrel{\text{def}}{=} \{X_n\}_{n\in\mathbb{N}}$ and $Y \stackrel{\text{def}}{=} \{Y_n\}_{n\in\mathbb{N}}$, be two polynomial-time constructible ensembles, and suppose that X and Y are indistinguishable in polynomial-time. Then X and Y are indistinguishable by polynomial-time sampling.*

An alternative formulation of Theorem 3.6 proceeds as follows. For every ensemble $Z \stackrel{\text{def}}{=} \{Z_n\}_{n\in\mathbb{N}}$ and every polynomial $m(\cdot)$ define the $m(\cdot)$-product of Z as the ensemble $\{(Z_n^{(1)}, ..., Z_n^{(m(n))})\}_{n\in\mathbb{N}}$, where the $Z_n^{(i)}$'s are independent copies of Z_n. Theorem 3.6 asserts that if the ensembles X and Y are polynomial-time indistinguishable, and each is polynomial-time constructible, then, for every polynomial $m(\cdot)$, the $m(\cdot)$-product of X and the $m(\cdot)$-product of X are polynomial-time indistinguishable.

The information theoretic analogue of the above theorem is quite obvious: if two ensembles are statistically close then also their polynomial-products must be statistically close (since the statistical difference between the m-products of two distributions is bounded by m times the distance between the individual distributions). Adapting the proof to the computational setting requires, as usual, a "reducibility argument". This argument uses, for the first time in this book, the *hybrid technique*. The hybrid technique plays a central role in demonstrating the computational indistinguishability of complex ensembles, constructed based on simpler (computational indistinguishable) ensembles. Subsequent application of the hybrid technique will involve more technicalities. Hence, the reader is urged not to skip the following proof.

Proof: The proof is by a "reducibility argument". We show that the existence of an efficient algorithm that distinguishes the ensembles X and Y using several samples, implies the existence of an efficient algorithm that distinguishes the ensembles X and Y using a single sample. The implication is proven using the following argument, which will be latter called a "hybrid argument".

Suppose, to the contradiction, that there is a probabilistic polynomial-time algorithm D, and polynomials $m(\cdot)$ and $p(\cdot)$, so that for infinitely many n's it holds that

$$\Delta(n) \stackrel{\text{def}}{=} |\text{Prob}\left(D(X_n^{(1)}, ..., X_n^{(m)})=1\right) - \text{Prob}\left(D(Y_n^{(1)}, ..., Y_n^{(m)})=1\right)| > \frac{1}{p(n)}$$

where $m \stackrel{\text{def}}{=} m(n)$, and the $X_n^{(i)}$'s and $Y_n^{(i)}$'s are as in Definition 3.4. In the sequel, we will derive a contradiction by presenting a probabilistic polynomial-time algorithm, D', that distinguishes the ensembles X and Y (in the sense of Definition 3.2).

For every k, $0 \leq k \leq m$, we define the *hybrid* random variable H_n^k as a (m-long) sequence consisting of k independent copies of X_n and $m-k$ independent copies of Y_n. Namely,

$$H_n^k \stackrel{\text{def}}{=} (X_n^{(1)}, ..., X_n^{(k)}, Y_n^{(k+1)}, ..., Y_n^{(m)})$$

where $X_n^{(1)}$ through $X_n^{(k)}$ and $Y_n^{(k+1)}$ through $Y_n^{(m)}$, are independent random variables with each $X_n^{(i)}$ identical to X_n and each $Y_n^{(i)}$ identical to Y_n. Clearly, $H_n^m = X_n^{(1)}, ..., X_n^{(m)}$, whereas $H_n^0 = Y_n^{(1)}, ..., Y_n^{(m)}$.

By our hypothesis, algorithm D can distinguish the extreme hybrids (i.e., H_n^0 and H_n^m). As the total number of hybrids is polynomial in n, a non-negligible gap between (the "accepting" probability of D on) the extreme hybrids translates into a non-negligible gap between (the "accepting" probability of D on) a pair of neighbouring hybrids. It follows that D, although not "designed to work on general hybrids", can distinguish a pair of neighbouring hybrids. The punch-line is that, algorithm D can be easily modified into an algorithm D' which distinguishes X and Y. Details follow.

We construct an algorithm D' which uses algorithm D as a subroutine. On input α (supposedly in the range of either X_n or Y_n), algorithm D' proceeds as follows. Algorithm D', first selects k uniformly in the set $\{0, 1, ..., m-1\}$. Using the efficient sampling algorithm for the ensemble X, algorithm D' generates k independent samples of X_n. These samples are denoted $x^1, ..., x^k$. Likewise, using the efficient sampling algorithm for the ensemble Y, algorithm D' generates $m-k-1$ independent samples of Y_n, denoted $y^{k+2}, ..., y^m$. Finally, algorithm D' invokes algorithm D and halts with output $D(x^1, ..., x^k, \alpha, y^{k+2}, ..., y^m)$.

Clearly, D' can be implemented in probabilistic polynomial-time. It is also easy to verify the following claims.

Claim 3.6.1:

$$\begin{aligned} \text{Prob}(D'(X_n)=1) &= \frac{1}{m}\sum_{k=0}^{m-1} \text{Prob}(D(H_n^{k+1})=1) \\ \text{and} & \\ \text{Prob}(D'(Y_n)=1) &= \frac{1}{m}\sum_{k=0}^{m-1} \text{Prob}(D(H_n^{k})=1) \end{aligned}$$

Proof: By construction of algorithm D', we have

$$D'(\alpha) = D(X_n^{(1)}, ..., X_n^{(k)}, \alpha, Y_n^{(k+2)}, ..., Y_n^{(m)})$$

Using the definition of the hybrids H_n^k, the claim follows. □

Claim 3.6.2:

$$|\text{Prob}(D'(X_n)=1) - \text{Prob}(D'(Y_n)=1)| = \frac{\Delta(n)}{m(n)}$$

Proof: Using Claim 3.6.1 for the first equality, we get

$$\begin{aligned}
&|\mathrm{Prob}(D'(X_n)=1) - \mathrm{Prob}(D'(Y_n)=1)| \\
&= \frac{1}{m} \cdot |\sum_{k=0}^{m-1} \mathrm{Prob}(D(H_n^{k+1})=1) - \mathrm{Prob}(D(H_n^k)=1)| \\
&= \frac{1}{m} \cdot |\mathrm{Prob}(D(H_n^m)=1) - \mathrm{Prob}(D(H_n^0)=1)| \\
&= \frac{\Delta(n)}{m}
\end{aligned}$$

The last equality follows by observing that $H_n^m = X_n^{(1)}, ..., X_n^{(m)}$ and $H_n^0 = Y_n^{(1)}, ..., Y_n^{(m)}$, and using the definition of $\Delta(n)$. □

Since by our hypothesis $\Delta(n) > \frac{1}{p(n)}$, for infinitely many n's, it follows that the probabilistic polynomial-time algorithm D' distinguishes X and Y in contradiction to the hypothesis of the theorem. Hence, the theorem follows. ■

It is worthwhile to give some thought to the *hybrid technique* (used for the first time in the above proof). The hybrid technique constitutes a special type of a "reducibility argument" in which the computational indistinguishability of *complex* ensembles is proven using the computational indistinguishability of *basic* ensembles. The actual reduction is in the other direction: efficiently distinguishing the basic ensembles is reduced to efficiently distinguishing the complex ensembles, and *hybrid* distributions are used in the reduction in an essential way. The following properties of the construction of the hybrids play an important role in the argument:

1. *Extreme hybrids collide with the complex ensembles*: this property is essential since what we want to prove (i.e., indistinguishability of the complex ensembles) relates to the complex ensembles.

2. *Neighbouring hybrids are easily related to the basic ensembles*: this property is essential since what we know (i.e., indistinguishability of the basic ensembles) relates to the basic ensembles. We need to be able to translate our knowledge (specifically computational indistinguishability) of the basic ensembles to knowledge (specifically computational indistinguishability) of any pair of neighbouring hybrids. Typically, it is required to efficiently transform strings in the range of a basic hybrid into strings in the range of a hybrid, so that the transformation maps the first basic distribution to one hybrid and the second basic distribution to the neighbouring hybrid. (In the proof of Theorem 3.6, the hypothesis that both X and Y are polynomial-time constructible is instrumental for such efficient transformation.)

3. *The number of hybrids is small* (i.e. polynomial): this property is essential in order to deduce the computational indistinguishability of extreme hybrids from the computational indistinguishability of neighbouring hybrids.

We remark that, in the course of an hybrid argument, a distinguishing algorithm referring to the complex ensembles is being analyzed and even executed on arbitrary hybrids. The reader may be annoyed of the fact that the algorithm "was not designed to work on such hybrids" (but rather only on the extreme hybrids). However, "an algorithm is an algorithm" and once it exists we can apply it to any input of our choice and analyze its performance on arbitrary input distributions.

3.2.4 Pseudorandom Ensembles

A special, yet important, case of computationally indistinguishable ensembles is the case in which one of the ensembles is uniform. Ensembles which are computational indistinguishable from the a uniform ensemble are called pseudorandom. Recall that U_m denotes a random variable uniformly distributed over the set of strings of length m. The ensemble $\{U_n\}_{n\in\mathbb{N}}$ is called the *standard uniform ensemble*. Yet, it will be convenient to call *uniform* also ensembles of the form $\{U_{l(n)}\}_{n\in\mathbb{N}}$, where l is a function on natural numbers.

Definition 3.7 (pseudorandom ensembles): *Let $U \stackrel{\text{def}}{=} \{U_{l(n)}\}_{n\in\mathbb{N}}$ be a uniform ensemble, and $X \stackrel{\text{def}}{=} \{X_n\}_{n\in\mathbb{N}}$ be an ensemble. The ensemble X is called* pseudorandom *if X and U are indistinguishable in polynomial-time.*

We stress that $|X_n|$ is not necessarily n (whereas $|U_m| = m$). In fact, with high probability $|X_n|$ equals $l(n)$).

In the above definition, as in the rest of this book, pseudorandomness is a shorthand for "pseudorandomness with respect to polynomial-time".

3.3 Definitions of Pseudorandom Generators

Pseudorandom ensembles, defined above, can be used instead of uniform ensemble in any efficient application without noticeable degradation in performance (otherwise the efficient application can be transformed into an efficient distinguisher of the supposedly-pseudorandom ensemble from the uniform one). Such a replacement is useful only if we can generate pseudorandom ensembles at a cheaper cost than required to generate a uniform ensemble. The cost of generating an ensemble has several aspects. Standard cost considerations are reflected by the time and space complexities. However, in the context of randomized algorithms, and

in particular in the context of generating probability ensembles, a major cost consideration is the quantity and quality of the randomness source used by the algorithm. In particular, in many applications (and especially in cryptography), *it is desirable to generate pseudorandom ensembles using as little randomness as possible.* This leads to the definition of a pseudorandom generator.

3.3.1 * A General Definition of Pseudorandom Generators

Definition 3.8 (pseudorandom generator): *A* pseudorandom generator *is a deterministic polynomial-time algorithm, G, satisfying the following two conditions:*

1. expansion: *for every $s \in \{0,1\}^*$ it holds that $|G(s)| > |s|$.*
2. pseudorandomness: *the ensemble $\{G(U_n)\}_{n\in\mathbb{N}}$ is pseudorandom.*

The input, s, to the generator is called its *seed.* It is required that a pseudorandom generator G always outputs a string longer than its seed, and that G's output, on a uniformly chosen seed, is pseudorandom. In other words, the output of a pseudorandom generator, on a uniformly chosen seed, must be polynomial-time indistinguishable from uniform, although it cannot be uniform (or even statistically close to uniform). To justify the last statement consider a uniform ensemble $\{U_{l(n)}\}_{n\in\mathbb{N}}$ that is polynomial-time indistinguishable from the ensemble $\{G(U_n)\}_{n\in\mathbb{N}}$ (such a uniform ensemble must exist by the pseudorandom property of G). We first claim that $l(n) > n$, since otherwise an algorithm that on input 1^n and a string α outputs 1 if and only if $|\alpha| > n$ will distinguish $G(U_n)$ from $U_{l(n)}$ (as $|G(U_n)| > n$ by the expansion property of G). It follows that $l(n) \geq n+1$. We next bound from below the statistical difference between $G(U_n)$ and $U_{l(n)}$, as follows

$$\begin{aligned}
\sum_x |\mathrm{Prob}(U_{l(n)}=x) - \mathrm{Prob}(G(U_n)=x)| &\geq \sum_{x\notin\{G(s):s\in\{0,1\}^n\}} |\mathrm{Prob}(U_{l(n)}=x) - \mathrm{Prob}(G(U_n)=x)| \\
&= (2^{l(n)} - 2^n)\cdot 2^{-l(n)} \\
&\geq \frac{1}{2}
\end{aligned}$$

It can be shown, see Exercise 8, that all the probability mass of $G(U_n)$, except for a negligible (in n) amount, is concentrated on strings of the same length and that this length equals $l(n)$, where $\{G(U_n)\}_{n\in\mathbb{N}}$ is polynomial-time indistinguishable from $\{U_{l(n)}\}_{n\in\mathbb{N}}$. For simplicity, we consider in the sequel, only pseudorandom generators G satisfying $|G(x)| = l(|x|)$ for all x's.

3.3.2 Standard Definition of Pseudorandom Generators

Definition 3.9 (pseudorandom generator - standard definition): *A* pseudorandom generator *is a deterministic polynomial-time algorithm, G, satisfying the following two conditions:*

1. expansion: *there exists a function $l : \mathbb{N} \mapsto \mathbb{N}$ so that $l(n) > n$ for all $n \in \mathbb{N}$, and $|G(s)| = l(|s|)$ for all $s \in \{0,1\}^*$.*
   The function l is called the expansion factor *of G.*
2. pseudorandomness (as above): *the ensemble $\{G(U_n)\}_{n\in\mathbb{N}}$ is pseudorandom.*

Again, we call the input to the generator a *seed*. The expansion condition requires that the algorithm G maps n-bit long seeds into $l(n)$-bit long strings, with $l(n) > n$. The pseudorandomness condition requires that the output distribution, induced by applying algorithm G to a uniformly chosen seed, is polynomial-time indistinguishable from uniform (although it is not statistically close to uniform - see justification in previous subsection).

The above definition says little about the expansion factor $l : \mathbb{N} \mapsto \mathbb{N}$. We merely know that for every n it holds that $l(n) \geq n + 1$, that $l(n) \leq \text{poly}(n)$, and that $l(n)$ can be computed in time polynomial in n. Clearly, a pseudorandom generator with expansion factor $l(n) = n + 1$ is of little value in practice, since it offers no significant saving in coin tosses. Fortunately, as shown in the subsequent subsection, even pseudorandom generators with such small expansion factor can be used to construct pseudorandom generators with any polynomial expansion factor. Hence, for every two expansion factors, $l_1 : \mathbb{N} \mapsto \mathbb{N}$ and $l_2 : \mathbb{N} \mapsto \mathbb{N}$, that can be computed in $\text{poly}(n)$-time, there exists a pseudorandom generator with expansion factor l_1 if and only if there exists a pseudorandom generator with expansion factor l_2. This statement is proven by using a pseudorandom generator with expansion factor $l_1(n) \stackrel{\text{def}}{=} n + 1$ to construct, for every polynomial $p(\cdot)$, a pseudorandom generator with expansion factor $p(n)$. Note that a pseudorandom generator with expansion factor $l_1(n) \stackrel{\text{def}}{=} n + 1$ can be derived from any pseudorandom generator (even from one in the general sense of Definition 3.8).

3.3.3 Increasing the Expansion Factor of Pseudorandom Generators

Given a pseudorandom generator, G_1, with expansion factor $l_1(n) = n + 1$, we construct a pseudorandom generator G with polynomial expansion factor, as follows.

Construction 3.10 *Let G_1 a deterministic polynomial-time algorithm mapping strings of length n into strings of length $n+1$, and let $p(\cdot)$ be a polynomial. Define $G(s) = \sigma_1 \cdots \sigma_{p(|s|)}$, where $s_0 \stackrel{\text{def}}{=} s$, the bit σ_i is the first bit of $G_1(s_{i-1})$, and s_i is the $|s|$-bit long suffix of $G_1(s_{i-1})$, for every $1 \leq i \leq p(|s|)$. (i.e., $\sigma_i s_i = G_1(s_{i-1})$)*

Hence, on input s, algorithm G applies G_1 for $p(|s|)$ times, each time on a new seed. Applying G_1 to the current seed yields a new seed (for the next iteration) and one extra bit (which is being output immediately). The seed in the first iteration is s itself. The seed in the i^{th} iteration is the $|s|$-long suffix of the string obtained from G_1 in the previous iteration. Algorithm G outputs the concatenation of the "extra bits" obtained in the $p(|s|)$ iterations. Clearly, G is polynomial-time computable and expands inputs of length n into output strings of length $p(n)$.

Theorem 3.11 *Let G_1, $p(\cdot)$, and G be as in Construction 3.10 (above). Then, if G_1 is a pseudorandom generator then so is G.*

Intuitively, the pseudorandomness of G follows from that of G_1 by replacing each application of G_1 by a random process which on input s outputs σs, where σ is uniformly chosen in $\{0,1\}$. Loosely speaking, the indistinguishability of a single application of the random process from a single application of G_1 implies that polynomially many applications of the random process are indistinguishable from polynomially many applications of G_1. The actual proof uses the hybrid technique.

Proof: The proof is by a "reducibility argument" . Suppose, to the contradiction, that G is not a pseudorandom generator. It follows that the ensembles $\{G(U_n)\}_{n\in\mathbb{N}}$ and $\{U_{p(n)}\}_{n\in\mathbb{N}}$ are not polynomial-time indistinguishable. We will show that it follows that the ensembles $\{G_1(U_n)\}_{n\in\mathbb{N}}$ and $\{U_{n+1}\}_{n\in\mathbb{N}}$ are not polynomial-time indistinguishable, in contradiction to the hypothesis that G_1 is a pseudorandom generator with expansion factor $l_1(n) = n+1$. The implication is proven, using the hybrid technique.

For every k, $0 \leq k \leq p(n)$, we define a hybrid $H^k_{p(n)}$ as follows. First we define, for every k, a function $g^k_n : \{0,1\}^n \mapsto \{0,1\}^k$ by letting $g^0_n(x) \stackrel{\text{def}}{=} \lambda$ (the empty string) and $g^{k+1}_n(x) = \sigma g^k_n(y)$, where σ is the first bit of $G_1(x)$ and y is the n-bit long suffix of $G_1(x)$ (i.e., $\sigma y = G_1(x)$). Namely, for every $k \leq p(|x|)$, the string $g^k_n(x)$ equals the k-bit long prefix of $G(x)$. Define the random variable $H^k_{p(n)}$ resulting by concatenating a uniformly chosen k-bit long string and the random variable $g^{p(n)-k}(U_n)$. Namely

$$H^k_{p(n)} \stackrel{\text{def}}{=} U^{(1)}_k g^{p(n)-k}(U^{(2)}_n)$$

where $U^{(1)}_k$ and $U^{(2)}_n$ are independent random variables (the first uniformly distributed over $\{0,1\}^k$ and the second uniformly distributed over $\{0,1\}^n$). Intuitively, the hybrid $H^k_{p(n)}$ consists of the k-bit long prefix of $U_{p(n)}$ and the $(p(n)-k)$-bit long suffix of $G(X_n)$, where X_n is obtained from U_n by applying G_1 for k times each time to the n-bit long suffix of the previous result. However, the later way of looking at the hybrids is less convenient for our purposes.

At this point it is clear that $H^0_{p(n)}$ equals $G(U_n)$, whereas $H^{p(n)}_{p(n)}$ equals $U_{p(n)}$. It follows that if an algorithm D can distinguish the extreme hybrids then D can also distinguish two neighbouring hybrids, since the total number of hybrids is polynomial in n and a non-negligible gap between the extreme hybrids translates into a non-negligible gap between some neighbouring hybrids. The punch-line is that, using the structure of neighbouring hybrids, algorithm D can be easily modified to distinguish the ensembles $\{G_1(U_n)\}_{n\in\mathbb{N}}$ and $\{U_{n+1}\}_{n\in\mathbb{N}}$. Details follow.

The core of the argument is the way in which the distinguishability of neighbouring hybrids relates to the distinguishability of $G(U_n)$ from U_{n+1}. As stated, this relation stems from the structure of neighbouring hybrids. Let us, thus, take a closer look at the hybrids $H^k_{p(n)}$ and $H^{k+1}_{p(n)}$, for some $0 \leq k \leq p(n)-1$. To this end, define a function $f^m : \{0,1\}^{n+1} \mapsto \{0,1\}^m$ by letting $f^0(z) \stackrel{\text{def}}{=} \lambda$ and $f^{m+1}(z) \stackrel{\text{def}}{=} \sigma g^m(y)$, where $z = \sigma y$ with $\sigma \in \{0,1\}$.

Claim 3.11.1:

1. $H^k_{p(n)} = U^{(1)}_k f^{p(n)-k}(X_{n+1})$, where $X_{n+1} = G_1(U^{(2)}_n)$.
2. $H^{k+1}_{p(n)} = U^{(1)}_k f^{p(n)-k}(Y_{n+1})$, where $Y_{n+1} = U^{(3)}_{n+1}$.

Proof:

1. By definition of the functions g^m and f^m, we have $g^m(x) = f^m(G_1(x))$. Using the definition of the hybrid $H^k_{p(n)}$, it follows that
$$H^k_{p(n)} = U^{(1)}_k g^{p(n)-k}(U^{(2)}_n) = U^{(1)}_k f^{p(n)-k}(G_1(U^{(2)}_n))$$
2. On the other hand, by definition $f^{m+1}(\sigma y) = \sigma g^m(y)$, and using the definition of the hybrid $H^{k+1}_{p(n)}$, we get
$$H^{k+1}_{p(n)} = U^{(1)}_{k+1} g^{p(n)-k-1}(U^{(2)}_n) = U^{(1)}_k f^{p(n)-k}(U^{(3)}_{n+1})$$

□

Hence distinguishing $G_1(U_n)$ from U_{n+1} is reduced to distinguishing the neighbouring hybrids (i.e. $H^k_{p(n)}$ and $H^{k+1}_{p(n)}$), by applying $f^{p(n)-k}$ to the input, padding the outcome (in front of) by a uniformly chosen string of length k, and applying the hybrid-distinguisher to the resulting string. Further details follow.

We assume, to the contrary of the theorem, that G is not a pseudorandom generators. Suppose that D is a probabilistic polynomial-time algorithm so that for some polynomial $q(\cdot)$ and for infinitely many n's it holds that
$$\Delta(n) \stackrel{\text{def}}{=} |\text{Prob}(D(G(U_n)=1) - \text{Prob}(D(U_{p(n)})=1)| > \frac{1}{q(n)}$$

We derive a contradiction by constructing a probabilistic polynomial-time algorithm, D', that distinguishes $G_1(U_n)$ from U_{n+1}.

Algorithm D' uses algorithm D as a subroutine. On input $\alpha \in \{0,1\}^{n+1}$, algorithm D' operates as follows. First, D' selects an integer k uniformly in the set $\{0,1,...,p(n)-1\}$, next D' selects β uniformly in $\{0,1\}^k$, and finally D' halts with output $D(\beta f^{p(n)-k}(\alpha))$, where $f^{p(n)-k}$ is as defined above.

Clearly, D' can be implemented in probabilistic polynomial-time (in particular $f^{p(n)-k}$ is computed by applying G_1 polynomially many times). It is left to analyze the performance of D' on each of the distributions $G_1(U_n)$ and U_{n+1}.

Claim 3.11.2:

$$\mathrm{Prob}(D'(G(U_n))=1) = \frac{1}{p(n)} \sum_{k=0}^{p(n)-1} \mathrm{Prob}(D(H^k_{p(n)})=1)$$

and

$$\mathrm{Prob}(D'(U_{n+1})=1) = \frac{1}{p(n)} \sum_{k=0}^{p(n)-1} \mathrm{Prob}(D(H^{k+1}_{p(n)})=1)$$

Proof: By construction of D' we get, for every $\alpha \in \{0,1\}^{n+1}$,

$$\mathrm{Prob}(D'(\alpha)=1) = \frac{1}{p(n)} \sum_{k=0}^{p(n)-1} \mathrm{Prob}(D(U_k f^{p(n)-k}(\alpha))=1)$$

Using Claim 3.11.1, our claim follows. □

Let $d^k(n)$ denote the probability that D outputs 1 on input taken from the hybrid $H^k_{p(n)}$ (i.e., $d^k(n) \stackrel{\mathrm{def}}{=} \mathrm{Prob}(D(H^k_{p(n)}=1))$. Recall that $H^0_{p(n)}$ equals $G(U_n)$, whereas $H^{p(n)}_{p(n)}$ equals $U_{p(n)}$. Hence, $d^0(n) = \mathrm{Prob}(D(G(U_n))=1)$, $d^{p(n)}(n) = \mathrm{Prob}(D(U_{p(n)})=1)$, and $\Delta(n) = |d^0(n) - d^{p(n)}(n)|$. Combining these facts with Claim 3.11.2, we get,

$$\begin{aligned} |\mathrm{Prob}(D'(G_1(U_n))=1) - \mathrm{Prob}(D'(U_{n+1})=1)| &= \frac{1}{p(n)} \cdot |\sum_{k=0}^{p(n)-1} d^k(n) - d^{k+1}(n)| \\ &= \frac{|d^0(n) - d^{p(n)}(n)|}{p(n)} \\ &= \frac{\Delta(n)}{p(n)} \end{aligned}$$

Recall that by our (contradiction) hypothesis $\Delta(n) > \frac{1}{q(n)}$, for infinitely many n's. Contradiction to the pseudorandomness of G_1 follows. ■

3.3.4 The Significance of Pseudorandom Generators

Pseudorandom generators have the remarkable property of being efficient "amplifiers/expanders of randomness". Using very little randomness (in form of a randomly chosen seed) they produce very long sequences which look random with respect to any efficient observer. Hence, the output of a pseudorandom generator may be used instead of "random sequences" in any efficient application requiring such (i.e., "random") sequences. The reason being that such an application may be viewed as a distinguisher. In other word, if some efficient algorithm suffers noticeable degradation in performance when replacing the random sequences it uses by pseudorandom one, then this algorithm can be easily modified into a distinguisher contradicting the pseudorandomness of the later sequences.

The generality of the notion of a pseudorandom generator is of great importance in practice. Once you are guaranteed that an algorithm is a pseudorandom generator you can use it in every efficient application requiring "random sequences" without testing the performance of the generator in the specific new application.

The benefits of pseudorandom generators to cryptography are innumerable (and only the most important ones will be presented in the subsequent chapters). The reason that pseudorandom generators are so useful in cryptography is that the implementation of all cryptographic tasks requires a lot of "high quality randomness". Thus, producing, exchanging and sharing large amounts of "high quality random bits" at low cost is of primary importance. Pseudorandom generators allow to produce (resp., exchange and/or share) $\mathrm{poly}(n)$ pseudorandom bits at the cost of producing (resp., exchanging and/or sharing) only n random bits!

A key property of pseudorandom sequences, that is used to justify the use of such sequences in cryptography, is the unpredictability of the sequence. Loosely speaking, a sequence is *unpredictable* if no efficient algorithm, given a prefix of the sequence, can guess its next bit with an advantage over one half that is not negligible. Namely,

Definition 3.12 (unpredictability): *An ensemble $\{X_n\}_{n\in\mathbf{N}}$ is called* unpredictable in polynomial-time *if for every probabilistic polynomial-time algorithm A and every polynomial $p(\cdot)$ and for all sufficiently large n's*

$$\mathrm{Prob}(A(1^n, X_n) = \mathrm{next}_A(1^n, X_n)) < \frac{1}{2} + \frac{1}{p(n)}$$

where $\mathrm{next}_A(x)$ returns the $i+1^{\mathrm{st}}$ bit of x if A on input $(1^n, x)$ reads only $i < |x|$ of the bits of x, and returns a uniformly chosen bit otherwise (i.e. in case A read the entire string x).

Clearly, pseudorandom ensembles are unpredictable in polynomial-time (see Exercise 14). It turns out that the converse holds as well. Namely, only pseudorandom ensembles are unpredictable in polynomial-time (see Exercise 15).

3.3.5 A Necessary Condition for the Existence of Pseudorandom Generators

Up to this point we have avoided the question of whether pseudorandom generators exist at all. Before saying anything positive, we remark that a necessary condition to the existence of pseudorandom generators is the existence of one-way function. Jumping ahead, we wish to reveal that this necessary condition is also sufficient: hence, pseudorandom generators exist if and only if one-way functions exist. At this point we only prove that the existence of pseudorandom generators implies the existence of one-way function. Namely,

Proposition 3.13 *Let G be a pseudorandom generator with expansion factor $l(n) = 2n$. Then the function $f:\{0,1\}^* \mapsto \{0,1\}^*$ defined by letting $f(x,y) \stackrel{\text{def}}{=} G(x)$, for every $|x| = |y|$, is a strongly one-way function.*

Proof: Clearly, f is polynomial-time computable. It is left to show that each probabilistic polynomial-time algorithm invert f with only negligible probability. We use a "reducibility argument". Suppose, on the contrary, that A is a probabilistic polynomial-time algorithm which for infinitely many n's inverts f on $f(U_{2n})$ with success probability at least $\frac{1}{\text{poly}(n)}$. We will construct a probabilistic polynomial-time algorithm, D, that distinguishes U_{2n} and $G(U_n)$ on these n's and reach a contradiction.

The distinguisher D uses the inverting algorithm A as a subroutine. On input $\alpha \in \{0,1\}^*$, algorithm D uses A in order to try to get a preimage of α under f. Algorithm D then checks whether the string it obtained from A is indeed a preimage and halts outputting 1 in case it is (otherwise it outputs 0). Namely, algorithm A computes $\beta \leftarrow A(\alpha)$, and outputs 1 if $f(\beta) = \alpha$ and 0 otherwise.

By our hypothesis, for some polynomial $p(\cdot)$ and infinitely many n's,

$$\text{Prob}(f(A(f(U_{2n}))) = f(U_{2n})) > \frac{1}{p(n)}$$

By f's construction the random variable $f(U_{2n})$ equals $G(U_n)$, and therefore $\text{Prob}(D(G(U_n)) = 1) > \frac{1}{p(n)}$. On the other hand, by f's construction at most 2^n different $2n$-bit long strings have a preimage under f. Hence, $\text{Prob}(f(A(U_{2n})) = U_{2n}) \leq 2^{-n}$. It follows that for infinitely many n's

$$|\text{Prob}(D(G(U_n)) = 1) - \text{Prob}(D(U_{2n}) = 1)| > \frac{1}{p(n)} - \frac{1}{2^n} > \frac{1}{2p(n)}$$

which contradicts the pseudorandomness of G. ■

3.4 Constructions based on One-Way Permutations

In this section we present constructions of pseudorandom generator based on one-way permutations. The first construction has a more abstract flavour, as it uses a single length preserving 1-1 one-way function (i.e., a single one-way permutation). The second construction utilizes the same underlying ideas to present practical pseudorandom generators based on collections of one-way permutations.

3.4.1 Construction based on a Single Permutation

By Theorem 3.11 (see Subsection 3.3.3), it suffices to present a pseudorandom generator expanding n-bit long seeds into $n+1$-bit long strings. Assuming that one-way permutations (i.e., 1-1 length preserving functions) exist, such pseudorandom generators can be constructed easily. We remind the reader that the existence of one-way permutation implies the existence of one-way permutation with corresponding hard-core predicates. Thus, it suffices to prove the following

Theorem 3.14 *Let f be a length-preserving 1-1 (strongly one-way) function, and let b be a hard-core predicate for f. Then the algorithm G, defined by $G(s) \stackrel{\text{def}}{=} f(s)b(s)$, is a pseudorandom generator.*

Intuitively, the ensemble $\{f(U_n)b(U_n)\}_{n\in\mathbb{N}}$ is pseudorandom since otherwise $b(U_n)$ can be efficiently predicted from $f(U_n)$. The proof merely formalizes this intuition.

Proof: We use a "reducibility argument". Suppose, on the contrary, that there exists an efficient algorithm D which distinguishes $G(U_n)$ from U_{n+1}. Recalling that $G(U_n) = f(U_n)b(U_n)$ and using the fact that f induces a permutation on $\{0,1\}^n$, we deduce that algorithm D distinguishes $f(U_n)b(U_n)$ from $f(U_n)U_1$. It follows that D distinguishes $f(U_n)b(U_n)$ from $f(U_n)\overline{b}(U_n)$, where $\overline{b}(x)$ is the complement bit of $b(x)$ (i.e., $\overline{b}(x) \stackrel{\text{def}}{=} \{0,1\}\text{–}b(x)$). Hence, algorithm D provides a good indication of $b(U_n)$ from $f(U_n)$, and can be easily modified into an algorithm guessing $b(U_n)$ from $f(U_n)$, in contradiction to the hypothesis that b is a hard-core predicate of f. Details follows.

We assume, on the contrary, that there exists a probabilistic polynomial-time algorithm D and a polynomial $p(\cdot)$ so that for infinitely many n's

$$|\text{Prob}(D(G(U_n))=1) - \text{Prob}(D(U_{n+1})=1)| > \frac{1}{p(n)}$$

Assume, without loss of generality, that for infinitely many n's it holds that

$$\Delta(n) \stackrel{\text{def}}{=} (\text{Prob}(D(G(U_n))=1) - \text{Prob}(D(U_{n+1})=1)) > \frac{1}{p(n)}$$

We construct a probabilistic polynomial-time algorithm, A, for predicting $b(x)$ from $f(x)$. Algorithm A uses the algorithm D as a subroutine. On input y (equals $f(x)$ for some x), algorithm A proceeds as follows. First, A selects uniformly $\sigma \in \{0,1\}$. Next, A applies D to $y\sigma$. Algorithm A halts outputting σ if $D(y\sigma) = 1$ and outputs the complement of σ, denoted $\overline{\sigma}$, otherwise.

Clearly, A works in polynomial-time. It is left to evaluate the success probability of algorithm A. We evaluate the success probability of A by considering two complementary events. The event we consider is whether or not "on input x algorithm A selects σ so that $\sigma = b(x)$".

Claim 3.14.1:

$$\begin{aligned}
\mathrm{Prob}(A(f(U_n)) = b(U_n) \mid \sigma = b(U_n)) &= \mathrm{Prob}(D(f(U_n)b(U_n)) = 1) \\
\mathrm{Prob}(A(f(U_n)) = b(U_n) \mid \sigma \neq b(U_n)) &= 1 - \mathrm{Prob}(D(f(U_n)\overline{b}(U_n)) = 1)
\end{aligned}$$

where $\overline{b}(x) = \{0,1\} - b(x)$.

Proof: By construction of A,

$$\begin{aligned}
\mathrm{Prob}(A(f(U_n)) = b(U_n) \mid \sigma = b(U_n)) &= \mathrm{Prob}(D(f(U_n)\sigma) = 1 \mid \sigma = b(U_n)) \\
&= \mathrm{Prob}(D(f(U_n)b(U_n)) = 1 \mid \sigma = b(U_n)) \\
&= \mathrm{Prob}(D(f(U_n)b(U_n)) = 1)
\end{aligned}$$

where the last equality follows since D's behavior is independent of the value of σ. Likewise,

$$\begin{aligned}
\mathrm{Prob}(A(f(U_n)) = b(U_n) \mid \sigma \neq b(U_n)) &= \mathrm{Prob}(D(f(U_n)\sigma) = 0 \mid \sigma = \overline{b}(U_n)) \\
&= \mathrm{Prob}(D(f(U_n)\overline{b}(U_n)) = 0 \mid \sigma = \overline{b}(U_n)) \\
&= 1 - \mathrm{Prob}(D(f(U_n)\overline{b}(U_n)) = 1)
\end{aligned}$$

The claim follows. □

Claim 3.14.2:

$$\begin{aligned}
\mathrm{Prob}(D(f(U_n)b(U_n)) = 1) &= \mathrm{Prob}(D(G(U_n)) = 1) \\
\mathrm{Prob}(D(f(U_n)\overline{b}(U_n)) = 1) &= 2 \cdot \mathrm{Prob}(D(U_{n+1}) = 1) - \mathrm{Prob}(D(f(U_n)b(U_n)) = 1)
\end{aligned}$$

Proof: By definition of G, we have $G(U_n) = f(U_n)b(U_n)$, and the first claim follows. To justify the second claim, we use the fact that f is a permutation over $\{0,1\}^n$, and hence $f(U_n)$ is uniformly distributed over $\{0,1\}^n$. It follows that U_{n+1} can be written as $f(U_n)U_1$. We get

$$\mathrm{Prob}(D(U_{n+1}) = 1) = \frac{\mathrm{Prob}(D(f(U_n)b(U_n)) = 1) + \mathrm{Prob}(D(f(U_n)\overline{b}(U_n)) = 1)}{2}$$

and the claim follows. □

Combining Claims 3.14.1 and 3.14.2, we get

$$\begin{aligned}
\mathrm{Prob}(A(f(U_n))=b(U_n)) &= \mathrm{Prob}(\sigma=b(U_n))\cdot\mathrm{Prob}(A(f(U_n))=b(U_n)\,|\,\sigma=b(U_n)) \\
&\quad +\mathrm{Prob}(\sigma\neq b(U_n))\cdot\mathrm{Prob}(A(f(U_n))=b(U_n)\,|\,\sigma\neq b(U_n)) \\
&= \frac{1}{2}\cdot\Big(\mathrm{Prob}(D(f(U_n)b(U_n))=1)+1-\mathrm{Prob}(D(f(U_n)\overline{b}(U_n))=1)\Big) \\
&= \frac{1}{2}+(\mathrm{Prob}(D(G(U_n))=1)-\mathrm{Prob}(D(U_{n+1})=1)) \\
&= \frac{1}{2}+\Delta(n)
\end{aligned}$$

Since $\Delta(n)>\frac{1}{p(n)}$ for infinitely many n's, we derive a contradiction and the theorem follows. ■

3.4.2 Construction based on Collections of Permutations

We now combine the underlying ideas of Construction 3.10 (of Subsection 3.3.3) and Theorem 3.14 (above) to present a construction of pseudorandom generators based on collections of one-way permutations. Let (I,D,F) be a triplet of algorithms defining a collection of one-way permutations (see Section 2.4.2). Recall that $I(1^n,r)$ denotes the output of algorithm I on input 1^n and coin tosses r. Likewise, $D(i,s)$ denotes the output of algorithm D on input i and coin tosses s. The reader may assume, for simplicity, that $|r|=|s|=n$. Actually, this assumption can be justified in general - see Exercise 13. However, in many applications it is more natural to assume that $|r|=|s|=q(n)$ for some fixed polynomial $q(\cdot)$. We remind the reader that Theorem 2.15 applies also to collections of one-way permutations.

Construction 3.15 *Let (I,D,F) be a triplet of algorithms defining a strong collection of one-way permutations, and let B be a hard-core predicate for this collection. Let $p(\cdot)$ be an arbitrary polynomial. Define $G(r,s)=\sigma_1\cdots\sigma_{p(n)}$, where $i\stackrel{\mathrm{def}}{=}I(1^n,r)$, $s_0\stackrel{\mathrm{def}}{=}D(i,s)$, and for every $1\leq j\leq p(|s|)$ it holds that $\sigma_j=B(s_{j-1})$ and $s_j=f_i(s_{j-1})$.*

On seed (r,s), algorithm G first uses r to determine a permutation f_i over D_i (i.e., $i\leftarrow I(1^n,r)$). Secondly, algorithm G uses s to determine a "starting point", s_0, in D_i. For simplicity, let us shorthand f_i by f. The essential part of algorithm G is the repeated application of the function f to the starting point s_0 and the extraction of a hard-core predicate for each resulting element. Namely, algorithm G computes a sequence of elements $s_1,...,s_{p(n)}$, where $s_j=f(s_{j-1})$ for every j (i.e., $s_j=f^{(j)}(s_0)$, where $f^{(j)}$ denotes j successive applications of the function f). Finally, algorithm G outputs the string $\sigma_1\cdots\sigma_{p(n)}$,

where $\sigma_j = B(s_{j-1})$. Note that σ_j is easily computed from s_{j-1} but is a "hard to approximate" from $s_j = f(s_{j-1})$. The pseudorandomness property of algorithm G depends on the fact that G does not output the intermediate s_j's. (In the sequel, we will see that outputting the last element, namely $s_{p(n)}$, does not hurt the pseudorandomness property.) The expansion property of algorithm G depends on the choice of the polynomial $p(\cdot)$. Namely, the polynomial $p(\cdot)$ should be larger than the polynomial $2q(\cdot)$ (where $2q(n)$ equals the total length of r and s corresponding to $I(1^n)$).

Theorem 3.16 *Let (I, D, F), B, $p(\cdot)$, and G be as in Construction 3.15 (above), so that $p(n) > 2q(n)$ for all n's. Suppose that for every i in the range of algorithm I, the random variable $D(i)$ is uniformly distbuted over the set D_i. Then G is a pseudorandom generator.*

Theorem 3.16 is an immediate corollary of the following proposition.

Proposition 3.17 *Let n and t be integers. For every i in the range of $I(1^n)$ and every x in D_i, define $G_{i,t}(x) = \sigma_1 \cdots \sigma_t$, where $s_0 \stackrel{\text{def}}{=} x$, $s_j = f_i^{(j)}(x)$ ($f^{(j)}$ denotes j successive applications of the function f) and $\sigma_j = B(s_{j-1})$, for every $1 \leq j \leq t$. Let (I, D, F) and B be as in Theorem 3.16 (above), I_n be a random variable representing $I(1^n)$, and $X_n = D(I_n)$ be a random variable depending on I_n. Then, for every polynomial $p(\cdot)$, the ensembles $\{(I_n, G_{I_n,p(n)}(X_n), f_{I_n}^{(p(n))}(X_n))\}_{n\in\mathbb{N}}$ and $\{(I_n, U_{p(n)}, f_{I_n}^{(p(n))}(X_n))\}_{n\in\mathbb{N}}$ are polynomial-time indistinguishable.*

Hence, the distinguishing algorithm gets, in addition to the $p(n)$-bit long sequence to be examined, also the index i chosen by G (in the first step of G's computation) and the last s_j (i.e., $s_{p(n)}$) computed by G. Even with this extra information it is infeasible to distinguish $G_{I_n,p(n)}(X_n) = G(1^n U_{2q(n)})$ from $U_{p(n)}$.

Proof Outline: The proof follows the proofs of Theorems 3.11 and 3.14 (of Subsection 3.3.3 and the current subsection, resp.). First, the statement is proven for $p(n) = 1$ (for all n's). This part is very similar to the proof of Theorem 3.14. Secondly, observe that the random variable X_n has distribution identical to the random variable $f_{I_n}(X_n)$, even conditioned on $I_n = i$ (of every i). Finally, assuming the validity of the case $p(\cdot) = 1$, the statement is proven for every polynomial $p(\cdot)$. This part is analogous to the proof of Theorem 3.11: one has to construct hybrids so that the k^{th} hybrid starts with an element i in the support of I_n, followed by k random bits, and ends with $G_{i,p(n)-k}(X_n)$ and $f_i^{p(n)-k}(X_n)$, where $X_n = D(i)$. The reader should be able to complete the argument. ■

Proposition 3.17 and Theorem 3.16 remain valid even if one relaxes the condition concerning the distribution of $D(i)$, and only requires that $D(i)$ is statistically close (as a function in $|i|$) to the uniform distribution over D_i.

3.4.3 Practical Constructions

As an immediate application of Construction 3.15, we derive pseudorandom generators based on either of the following assumptions

- *The Intractability of the Discrete Logarithm Problem*: The genertor is based on the fact that it is hard to predict, given a prime P, a primitive element G in the multiplicative group mod P, and an element Y of the group, whether there exists $0 \leq x \leq P/2$ so that $Y \equiv G^x \bmod P$. In other words, this bit constitues a hard-core for the DLP collection (of Subsection 2.4.3).
- *The difficulty of inverting RSA*: The genertor is based on the fact that the least significant bit constitues a hard-core for the RSA collection.
- *The Intractability of Factoring Blum Integers*: The genertor is based on the fact that the least significant bit constitues a hard-core for the Rabin collection, when viewed as a collection of permutations over the quadratic residues of Blum integers (see Subsection 2.4.3).

We ellaborate on the last example since it offers the most efficient implementation and yet is secure under a widely believed intractability assumption. The generator uses its seed in order to generate a composite number, N, which is the product of two relatively large primes.

*** PROVIDE DETAILS ABOVE. MORE EFFICIEN HEURISTIC BELOW...

3.5 * Construction based on One-Way Functions

It is known that one-way functions exist if and only if pseudorandom generators exist. However, the known construction which transforms arbitrary one-way functions into pseudorandom generators is impractical. Furthermore, the proof that this construction indeed yields pseudorandom generators is very complex and unsuitable for a book of the current nature. Instead, we refrain to present some of the ideas underlying this construction.

3.5.1 Using 1-1 One-Way Functions

Recall that if f is a 1-1 length-preserving one-way function and b is a corresponding hard-core predicate then $G(s) \stackrel{\text{def}}{=} f(s)b(s)$ constitutes a pseudorandom generator. Let us relax the condition imposed on f and assume that f is a 1-1 one-way function (but is not necessarily length preserving). Without loss of generality, we may assume that there exists a polynomial $p(\cdot)$ so that $|f(x)| = p(|x|)$ for all x's. In case f is not length preserving, it follows that

$p(n) > n$. At first glance, one may think that we only benefit in such a case since f by itself has an expanding property. The impression is misleading since the expanded strings may not "look random". In particular, it may be the case that the first bit of $f(x)$ is zero for all x's. More generally, $f(U_n)$ may be easy to distinguish from $U_{p(n)}$ (otherwise f itself constitutes a pseudorandom generator). Hence, in the general case, we need to get rid of the expansion property of f since it is not accompanied by a "pseudorandom" property. In general, we need to shrink $f(U_n)$ back to length $\leq n$ so that the shrunk result induces uniform distribution. The question is how to *efficiently* carry on this process (i.e., of shrinking $f(x)$ back to length $|x|$, so that the shrunk $f(U_n)$ induces a uniform distribution on $\{0,1\}^n$).

Suppose that there exists an efficiently computable function h so that $f_h(x) \stackrel{\text{def}}{=} h(f(x))$ is length preserving and 1-1. In such a case we can let $G(s) \stackrel{\text{def}}{=} h(f(s))b(s)$, where b is a hard-core predicate for f, and get a pseudorandom generator. The pseudorandomness of G follows from the observation that if b is a hard-core for f it is also a hard-core for f_h (since an algorithm guessing $b(x)$ from $h(f(x))$ can be easily modified so that it guesses $b(x)$ from $f(x)$, by applying h first). The problem is that we "know nothing about the structure" of f and hence are not guaranteed that h as above does exist. An important observation is that a uniformly selected *hashing* function will have approximately the desired properties. Hence, hashing functions play a central role in the construction, and consequently we need to discuss these functions first.

Hashing Functions

The following terminology relating to hashing functions is merely an ad-hoc terminology (which is not a standard one). Let S_n^m be a set of strings representing functions mapping n-bit strings to m-bit strings. In the sequel we freely associate the strings in S_n^m with the functions that they represent. Let H_n^m be a random variable uniformly distributed over the set S_n^m. We call S_n^m a hashing family if it satisfies the following three conditions:

1. S_n^m is *a pairwise independent family of mappings*: for every $x \neq y \in \{0,1\}^n$, the random variables $H_n^m(x)$ and $H_n^m(y)$ are independent and uniformly distributed in $\{0,1\}^m$.
2. S_n^m has *succinct representation*: $S_n^m = \{0,1\}^{\text{poly}(n \cdot m)}$.
3. S_n^m can be *efficiently evaluated*: there exists a polynomial-time algorithm that, on input a representation of a function, h (in S_n^m), and a string $x \in \{0,1\}^n$, returns $h(x)$.

A widely used hashing family is the set of affine transformations mapping n-dimensional binary vectors to m-dimensional ones (i.e., transformations affected by multiplying the n-dimensional vector by an n-by-m binary matrix and adding an m-dimensional vector to

the result). A hashing family with more succinct representation is obtained by considering only the transformations affected by Toeplitz matrices (i.e., matrices which are invariant along the diagonals). For further details see Exercise 16. Following is a lemma, concerning hashing functions, that is central to our application (as well as to many applications of hashing functions in complexity theory). Loosely speaking, the lemma asserts that most h's in a hashing family have $h(X_n)$ distributed almost uniformly, provided X_n does not assign too much probability mass to any single string.

Lemma 3.18 *Let $m < n$ be integers, S_n^m be a hashing family, and b and δ be two reals so that $b \leq n$ and $\delta \geq 2^{-\frac{b-m}{2}}$. Suppose that X_n is a random variable distributed over $\{0,1\}^n$ so that for every x it holds that $\mathrm{Prob}(X_n = x) \leq 2^{-b}$. Then, for every $\alpha \in \{0,1\}^m$, and for all but a $2^{-(b-m)}\delta^{-2}$ fraction of the h's in S_n^m, it holds that*

$$\mathrm{Prob}(h(X_n) = \alpha) \in (1 \pm \delta) \cdot 2^{-m}$$

A function h not satisfying $\mathrm{Prob}(h(X_n) = \alpha) \in (1 \pm \delta) \cdot 2^{-m}$ is called *bad* (for α and the random variable X_n). Averaging on all h's we have $\mathrm{Prob}(h(X_n) = \alpha)$ equal 2^{-m}. Hence the lemma bounds the fraction of h's which deviate from the average value. Typically we shall use $\delta \stackrel{\mathrm{def}}{=} 2^{-\frac{b-m}{3}} \ll 1$ (making the deviation from average equal the fraction of bad h's). Another useful choice is $\delta > 1$ (which yields an even smaller fraction of bad h's, yet badness has only a "lower bound interpretation", i.e. $\mathrm{Prob}(h(X_n) = \alpha) \leq (1 + \delta) \cdot 2^{-m}$).

Proof: Fix an arbitrary random variable X_n, satisfying the conditions of the lemma, and an arbitrary $\alpha \in \{0,1\}^m$. Denote $w_x \stackrel{\mathrm{def}}{=} \mathrm{Prob}(X_n = x)$. For every h we have

$$\mathrm{Prob}(h(X_n) = \alpha) = \sum_x w_x \zeta_x(h)$$

where $\zeta_x(h)$ equal 1 if $h(x) = \alpha$ and 0 otherwise. Hence, we are interested in the probability, taken over all possible choices of h, that $|2^{-m} - \sum_x w_x \zeta_x(h)| > \delta 2^{-m}$. Looking at the ζ_x's as random variables defined over the random variable H_n^m, it is left to show that

$$\mathrm{Prob}\left(|2^{-m} - \sum_x w_x \zeta_x| > \delta \cdot 2^{-m} \right) > \frac{2^{-(b-m)}}{\delta^2}$$

This is proven by applying Chebyshev's Inequality, using the fact that the ζ_x's are pairwise independent, and that ζ_x equals 1 with probability 2^{-m} (and 0 otherwise). (We also take advantage on the fact that $w_x \leq 2^{-b}$.) Namely,

$$\begin{aligned} \mathrm{Prob}\left(|2^{-m} - \sum_x w_x \zeta_x| > \delta \cdot 2^{-m} \right) &\leq \frac{\mathrm{Var}\left(\sum_x w_x \zeta_x\right)}{(\delta \cdot 2^{-m})^2} \\ &< \frac{\sum_x 2^{-m} w_x^2}{\delta^2 \cdot 2^{-2m}} \\ &\leq \frac{2^{-m} 2^{-b}}{\delta^2 \cdot 2^{-2m}} \end{aligned}$$

The lemma follows. ■

Constructing "Almost" Pseudorandom Generators

Using any 1-1 one-way function and any hashing family, we can take a major step towards constructing a pseudorandom generator.

Construction 3.19 *Let $f:\{0,1\}^* \mapsto \{0,1\}^*$ be a function satisfying $|f(x)| = p(|x|)$ for some polynomial $p(\cdot)$ and all x's. For any integer function $l:\mathbb{N} \mapsto \mathbb{N}$, let $g:\{0,1\}^* \mapsto \{0,1\}^*$ be a function satisfying $|g(x)| = l(|x|)+1$, and $S_{p(n)}^{n-l(n)}$ be a hashing family. For every $x \in \{0,1\}^n$ and $h \in S_{p(n)}^{n-l(n)}$, define*

$$G(x,h) \stackrel{\text{def}}{=} (h(f(x)), h, g(x))$$

Clearly, $|G(x,h)| = (|x| - l(|x|)) + |h| + (l(|x|)+1) = |x| + |h| + 1$.

Proposition 3.20 *Let f, l, g and G be as above. Suppose that f is 1-to-1 and g is a hard-core function of f. Then, for every probabilistic polynomial-time algorithm A, every polynomial $p(\cdot)$, and all sufficiently large n's*

$$|\text{Prob}(A(G(U_n, U_k)) = 1) - \text{Prob}(A(U_{n+k+1}) = 1)| < 2^{-\frac{l(n)}{3}} + \frac{1}{p(n)}$$

where k is the length of the representation of the hashing functions in $S_{p(n)}^{n-l(n)}$.

The proposition can be extended to the case in which the function f is polynomial-to-1 (instead of 1-to-1). Specifically, let f satisfy $|f^{-1}f(x)| < q(|x|)$, for some polynomial $q(\cdot)$ and all sufficiently long x's. The modified proposition asserts that for every probabilistic polynomial-time algorithm A, every polynomial $p(\cdot)$, and all sufficiently large n's

$$|\text{Prob}(A(G(U_n, U_k)) = 1) - \text{Prob}(A(U_{n+k+1}) = 1)| < 2^{-\frac{l(n) - \log_2 q(n)}{3}} + \frac{1}{p(n)}$$

where k is as above.

In particular, the above proposition holds for functions $l(\cdot)$ of the form $l(n) \stackrel{\text{def}}{=} c \log_2 n$, where $c > 0$ is a constant. For such functions l, every one-way function (can be easily modified into a function which) has a hard-core g as required in the proposition's hypothesis (see Subsection 2.5.3). Hence, we get very close to constructing a pseudorandom generator.

Proof Sketch: We first note that

$$\begin{aligned} G(U_n U_k) &= (H_{p(n)}^{n-l(n)}(f(U_n)), H_{p(n)}^{n-l(n)}, g(U_n)) \\ U_{n+k+1} &= (U_{n-l(n)}, H_{p(n)}^{n-l(n)}, U_{l(n)+1}) \end{aligned}$$

We consider the hybrid $(H_{p(n)}^{n-l(n)}(f(U_n)), H_{p(n)}^{n-l(n)}, U_{l(n)+1})$. The proposition is a direct consequence of the following two claims.

Claim 3.20.1: The ensembles

$$\{(H_{p(n)}^{n-l(n)}(f(U_n)), H_{p(n)}^{n-l(n)}, g(U_n))\}_{n\in\mathbb{N}}$$

and

$$\{(H_{p(n)}^{n-l(n)}(f(U_n)), H_{p(n)}^{n-l(n)}, U_{l(n)+1})\}_{n\in\mathbb{N}}$$

are polynomial-time indistinguishable.

Proof Idea: Use a "reducibility argument". If the claim does not hold then contradiction to the hypothesis that g is a hard-core of f is derived. □

Claim 3.20.2: The statistical difference between the random variables

$$(H_{p(n)}^{n-l(n)}(f(U_n)), H_{p(n)}^{n-l(n)}, U_{l(n)+1})$$

and

$$(U_{n-l(n)}, H_{p(n)}^{n-l(n)}, U_{l(n)+1})$$

is bounded by $2^{-l(n)/3}$.

Proof Idea: Use the hypothesis that $S_{p(n)}^{n-l(n)}$ is a hashing family, and apply Lemma 3.18. □

Since the statistical difference is a bound on the ability of algorithms to distinguish, the proposition follows. ■

Applying Proposition 3.20

Once the proposition is proven we consider the possibilities of applying it in order to construct pseudorandom generators. We stress that applying Proposition 3.20, with length function $l(\cdot)$, requires having a hard-core function g for f with $|g(x)| = l(|x|) + 1$. By Theorem 2.17 (of Subsection 2.5.3) such hard-core functions exist practically for all one-way functions, provided that $l(\cdot)$ is logarithmic (actually, Theorem 2.17 asserts that such hard-cores exist for a modification of any one-way function which preserves its 1-1 property). Hence, combining Theorem 2.17 and Proposition 3.20, and using a logarithmic length function, we get very close to constructing a pseudorandom generator. In particular, for every polynomial $p(\cdot)$, using $l(n) \stackrel{\text{def}}{=} 3\log_2 p(n)$, we can construct a deterministic polynomial-time

algorithm expanding n-bit long seeds into $(n+1)$-bit long strings so that no polynomial-time algorithm can distinguish the output strings from uniformly chosen ones, with probability greater than $\frac{1}{p(n)}$ (except for finitely many n's). Yet, this does not imply that the output is pseudorandom (i.e., that the distinguishing gap is smaller than any polynomial fraction). A final trick is needed (since we cannot use $l(\cdot)$ bigger than any logarithmic function). In the sequel we present two alternative ways for obtaining a pseudorandom generator from the above construction.

The first alternative is to use Construction 3.10 (of Subsection 3.10) in order to increase the expansion factor of the above algorithms. In particular, for every integer k, we construct a deterministic polynomial-time algorithm expanding n-bit long seeds into n^3-bit long strings so that no polynomial-time algorithm can distinguish the output strings from uniformly chosen ones, with probability greater than $\frac{1}{n^k}$ (except for finitely many n's). Denote these algorithms by $G_1, G_2, ...$, and construct a pseudorandom generator G by letting

$$G(s) \stackrel{\text{def}}{=} G_1(s_1) \oplus G_2(s_2) \cdots \oplus G_{k(|s|)}(s_{k(|s|)})$$

where $\oplus$ denotes bit-by-bit exclusive-or of strings, $s_1 s_2 \cdots s_{k(|s|)} = s$, $|s_i| = \frac{|s|}{k(|s|)} \pm 1$, and $k(n) \stackrel{\text{def}}{=} \sqrt[3]{n}$. Clearly, $|G(s)| \approx k(|s|) \cdot (\frac{|s|}{k(|s|)})^3 = |s|^2$. The pseudorandomness of G follows by a "reducibility argument". (The choice of the function k is rather arbitrary, and any unbounded function $k(\cdot)$ satisfying $k(n) < n^{2/3}$ will do.)

The second alternative is to apply Construction 3.19 to the function $\overline{f}$ defined by

$$\overline{f}(x_1, ..., x_n) \stackrel{\text{def}}{=} f(x_1) \cdots f(x_n)$$

where $|x_1| = \cdots = |x_n| = n$. The benefit in applying Construction 3.19 to the function $\overline{f}$ is that we can use $l(n^2) \stackrel{\text{def}}{=} n - 1$, and hence Proposition 3.20 yields that G is a pseudorandom generator. All that is left is to show that $\overline{f}$ has a hard core function which maps n^2-bit strings into n-bit strings. Assuming that b is a hard-core predicate of the function f, we can construct such a hard-core function for $\overline{f}$. Specifically,

Construction 3.21 *Let $f : \{0,1\}^* \mapsto \{0,1\}^*$ and $b : \{0,1\}^* \mapsto \{0,1\}$. Define*

$$\begin{aligned} \overline{f}(x_1, ..., x_n) &\stackrel{\text{def}}{=} f(x_1) \cdots f(x_n) \\ \overline{g}(x_1, ..., x_n) &\stackrel{\text{def}}{=} b(x_1) \cdots b(x_n) \end{aligned}$$

where $|x_1| = \cdots = |x_n| = n$.

Proposition 3.22 *Let f and b be as above. If b is a hard-core predicate of f then $\overline{g}$ is a hard-core function of $\overline{f}$.*

Proof Idea: Use the hybrid technique. The i^{th} hybrid is

$$\left(\overline{f}(U_n^{(1)},...,U_n^{(n)}),b(U_n^{(1)}),...,b(U_n^{(i)}),U_1^{(i+1)},...,U_1^{(n)}\right)$$

Use a reducibility argument (as in Theorem 3.14 of Subsection 3.4.1) to convert a distinguishing algorithm into one predicting b from f. ■

Using either of the above alternatives, we get

Theorem 3.23 *If there exist 1-1 one-way functions then pseudorandom generators exist as well.*

The entire argument can be extended to the case in which the function f is polynomial-to-1 (instead of 1-to-1). Specifically, let f satisfy $|f^{-1}f(x)| < q(|x|)$, for some polynomial $q(\cdot)$ and all sufficiently long x's. Then if f is one-way then (either of the above alternatives yields that) pseudorandom generators exists. Proving the statement using the first alternative is quite straightforward given the discussion proceeding Proposition 3.20. In proving the statement using the second alternative apply Construction 3.19 to the function $\overline{f}$ with $l(n^2) \stackrel{\text{def}}{=} n\cdot(1+\log_2 q(n))-1$. This requires showing that $\overline{f}$ has a hard core function which maps n^2-bit strings into $n(1+\log_2 q(n))$-bit strings. Assuming that g is a hard-core function of the function f, with $|g(x)| = 1+\log_2 q(|x|)$, we can construct such a hard-core function for $\overline{f}$. Specifically,

$$\overline{g}(x_1,...,x_n) \stackrel{\text{def}}{=} g(x_1)\cdots g(x_n)$$

where $|x_1| = \cdots = |x_n| = n$.

3.5.2 Using Regular One-Way Functions

The validity of Proposition 3.20 relies heavily on the fact that if f is 1-1 then $f(U_n)$ maintains the "entropy" of U_n in a strong sense (i.e., $\text{Prob}(f(U_n)=\alpha) \leq 2^{-n}$ for every α). In this case, it was possible to shrink $f(U_n)$ and get almost uniform distribution over $\{0,1\}^{n-l(n)}$. As stressed above, the condition may be relaxed to requiring that f is polynomial-to-1 (instead of 1-to-1). In such a case only logarithmic loss of "entropy" occurs, and such a loss can be compensated by an appropriate increase in the range of the hard-core function. We stress that hard-core functions of logarithmic length (i.e., satisfying $|g(x)| = O(\log|x|)$) can be constructed for any one-way function. However, in general, the function f may not be polynomial-to-1 and in particular it can map exponentially many images to the same range element. If this is the case then applying f to U_n yields a great loss in "entropy", which cannot be compensated using the above methods. For example, if $f(x,y) \stackrel{\text{def}}{=} f'(x)0^{|y|}$, for $|x|=|y|$, then $\text{Prob}(f(U_n)=\alpha) \geq 2^{-\frac{|\alpha|}{2}}$ for some α's. In this case, achieving uniform distribution from $f(U_n)$ requires shrinking it to length $\approx n/2$. In general, we cannot compensate

for these lost bits since f may not have a hard-core with such huge range (i.e., a hard-core g satisfying $|g(\alpha)| = \frac{|\alpha|}{2}$). Hence, in this case, the above methods fail for constructing an algorithm that expands its input into a longer output. A new idea is needed, and indeed presented below.

The idea is that, in case f maps different preimages into the same image y, we can augment y by the index of the preimage, in the set $f^{-1}(y)$, *without damaging the hardness-to-invert of* f. Namely, we define $F(x) \stackrel{\text{def}}{=} f(x) \cdot \mathrm{idx}_f(x)$, where $\mathrm{idx}_f(x)$ denotes the index (say by lexicographic order) of x in the set $\{x' : f(x') = f(x)\}$. We claim that inverting F is not substantially easier than inverting f. This claim can be proven by a "reducibility argument". Given an algorithm for inverting F we can invert f as follows. On input y (supposedly in the range of $f(U_n)$), we first select m uniformly in $\{1, ..., n\}$, next select i uniformly in $\{1, ..., 2^m\}$, and finally try to invert F on (y, i). When analyzing this algorithm, consider the case $i = \lceil \log_2 |f^{-1}(y)| \rceil$.

The function F suggested above does preserve the hardness-to-invert of f. The problem is that it does not preserve the easy-to-compute property of f. In particular, for general f it is not clear how to compute $\mathrm{idx}_f(x)$ (i.e., the best we can say is that this task can be performed in polynomial *space*). Again, hashing functions come to the rescue. Suppose, for example that f is 2^m-to-1 on strings of length n. Then, we can set $\mathrm{idx}_f(x) = (H_n^m, H_n^m(x))$, obtaining "probabilistic indexing" of the set of preimages. We stress that applying the above trick requires having a good estimate for the size of the set of preimages (of a given image). That is, given x it should be easy to compute $|f^{-1}f(x)|$. A simple case where such an estimate can be handy is the case of regular functions.

Definition 3.24 (Regular functions): *A function $f : \{0,1\}^* \mapsto \{0,1\}^*$ is called* regular *if there exists an integer function $m : \mathbb{N} \mapsto \mathbb{N}$ so that for all sufficiently long $x \in \{0,1\}^*$ it holds*

$$|\{y : f(x) = f(y) \wedge |x| = |y|\}| = 2^{m(|x|)}$$

For simplicity, the reader may further assume that there exists an algorithm that on input n computes $m(n)$ in $\mathrm{poly}(n)$-time. As we shall see, in the end of this subsection, one can do without this assumption. For sake of simplicity (of notation), we assume in the sequel that if $f(x) = f(y)$ then $|x| = |y|$.

Construction 3.25 *Let $f : \{0,1\}^* \mapsto \{0,1\}^*$ be a regular function with $m(|x|) = \log_2 |f^{-1}f(x)|$ for some integer function $m(\cdot)$. Let $l : \mathbb{N} \mapsto \mathbb{N}$ be an integer function, and $S_n^{m(n)-l(n)}$ be a hashing family. For every $x \in \{0,1\}^n$ and $h \in S_n^{m(n)-l(n)}$, define*

$$F(x, h) \stackrel{\text{def}}{=} (f(x), h(x), h)$$

If f can be computed in polynomial-time and $m(n)$ can be computed from n in poly(n)-time, then F can be computed in polynomial-time. We now show that if f is a regular one-way function, then F is "hard to invert". Furthermore, if $l(\cdot)$ is logarithmic then F is "almost 1-1".

Proposition 3.26 *Let f, m, l and F be as above. Suppose that there exists an algorithm that on input n computes $m(n)$ in* poly(n)*-time. Then,*

1. *F* is "almost" 1-1:

$$\text{Prob}\left(|F^{-1}F(U_n, H_n^{m(n)-l(n)})| > 2^{l(n)+1}\right) < 2^{-\frac{l(n)}{2}}$$

   (Recall that H_n^k denotes a random variable uniformly distributed over S_n^k.)

2. *F* "preserves" the one-wayness of f:
   If f is strongly (resp. weakly) one-way then so is F.

Proof Sketch: Part (1) is proven by applying Lemma 3.18, using the hypothesis that $S_n^{m(n)-l(n)}$ is a hashing family. Part (2) is proven using a "reducibility argument" . Assuming, to the contradiction, that there exists an efficient algorithm A that inverts F with unallowable success probability, we construct an efficient algorithm A' that inverts f with unallowable success probability (reaching contradiction). For sake of concreteness, we consider the case in which f is strongly one-way, and assume to the contradiction that algorithm A inverts F on $F(U_n, H_n^{m(n)-l(n)})$ with success probability $\epsilon(n)$, so that $\epsilon(n) > \frac{1}{\text{poly}(n)}$ for infinitely many n's. Following is a description of A'.

On input y (supposedly in the range of $f(U_n)$), algorithm A' repeats the following experiment for poly$(\frac{n}{\epsilon(n)})$ many times. Algorithm A' selects uniformly $h \in S_n^{m(n)-l(n)}$ and $\alpha \in \{0,1\}^{m(n)-l(n)}$, and initiates A on input (y, α, h). Algorithm A' sets x to be the n-bit long prefix of $A(y, \alpha, h)$, and outputs x if $y = f(x)$. Otherwise, algorithm A' continues to the next experiment.

Clearly, algorithm A' runs in polynomial-time, provided that $\epsilon(n) > \frac{1}{\text{poly}(n)}$. We now evaluate the success probability of A'. For every possible input, y, to algorithm A', we consider a random variable X_n uniformly distributed in $f^{-1}(y)$. Let $\delta(y)$ denote the success probability of algorithm A on input $(y, H_n^k(X_n), H_n^k)$, where $n \stackrel{\text{def}}{=} |y|$ and $k \stackrel{\text{def}}{=} m(n) - l(n)$. Clearly, $\text{Exp}(\delta(f(U_n))) = \epsilon(n)$, and $\text{Prob}(\delta(f(U_n)) > \frac{\epsilon(n)}{2}) > \frac{\epsilon(n)}{2}$ follows. We fix an arbitrary $y \in \{0,1\}^n$ so that $\delta(y) > \frac{\epsilon(n)}{2}$. We prove the following technical claim.

Claim 3.26.1: Let n, k and X_n be as above. Suppose that B is a set of pairs, and

$$\delta \stackrel{\text{def}}{=} \text{Prob}((H_n^k(X_n), H_n^k) \in B)$$

Then,

$$\text{Prob}((U_k, H_n^k) \in B) > \frac{\delta^4}{2^8 \cdot k}$$

Using this claim, it follows that the probability that A' inverts f on y in a single iteration is at least $(\frac{\delta(y)}{4})^4 \cdot \frac{1}{k}$. We reach a contradiction (to the one-wayness of f), and the proposition follows. All that is left is to prove Claim 3.26.1. The proof, given below, is rather technical. ■

We stress that the fact that $m(n)$ can be computed from n does not play an essential role in the reducibility argument (as it is possible to try all possible values of $m(n)$).

Claim 3.26.1 is of interest for its own sake. However, its proof provides no significant insights and may be skipped without significant damage (especially by readers that are more interested in cryptography than in "probabilistic analysis").

Proof of Claim 3.26.1: We first use Lemma 3.18 to show that only a "tiny" fraction of the hashing functions in S_n^k can map "large" probability mass into "small" subsets. Once this is done, the claim is proven by dismissing those few bad functions and relating the two probabilities, appearing in the statement of the claim, conditioned on the function not being bad. Details follow.

We begin by bounding the fraction of the hashing functions that map "large" probability mass into "small" subsets. We say that a function $h \in S_n^k$ is (T, Δ)-*expanding* if there exists a set $R \subset \{0,1\}^k$ of cardinality $\Delta \cdot 2^k$ so that $\text{Prob}(h(X_n) \in R)) \geq (T+1) \cdot \Delta$. In other words, h maps to some set of density Δ a probability mass $T+1$ times the density of the set. Our first goal is to prove that at most $\frac{\delta}{4}$ of the h's are $(\frac{32k}{\delta^2}, \frac{\delta^3}{64k})$-expanding. In other words, only $\frac{\delta}{4}$ of the function map to some set of density $\frac{\delta^3}{64k}$ a probability mass of more than $\frac{\delta}{2}$.

We start with a related question. We say that $\alpha \in \{0,1\}^k$ is *t-overweighted* by the function h if $\text{Prob}(h(X_n) = \alpha)) \geq (t+1) \cdot 2^{-k}$. A function $h \in S_n^k$ is called (t, ρ)-*overweighting* if there exists a set $R \subset \{0,1\}^k$ of cardinality $\rho 2^k$ so that each $\alpha \in R$ is t-overweighted by h. (Clearly, if h is (t, ρ)-overweighting then it is also (t, ρ)-expanding, but the converse is not necessarily true.) We first show that at most a $\frac{1}{t^2 \rho}$ fraction of the h's are (t, ρ)-overweighting. The proof is given in the rest of this paragraph. Recall that $\text{Prob}(X_n = x) \leq 2^{-k}$, for every x. Using Lemma 3.18, it follows that each $\alpha \in \{0,1\}^k$ is t-overweighted by at most a t^{-2} fraction of the h's. Assuming, to the contradiction, that more than a $\frac{1}{t^2 \rho}$ fraction of the h's are (t, ρ)-overweighting, we construct a bipartite graph by connecting each of these h's with the α's that it t-overweights. Contradiction follows by observing that there exists an α which is connected to more than $\frac{\frac{|S_n^k|}{t^2 \rho} \cdot \rho 2^k}{2^k} = \frac{1}{t^2} \cdot |S_n^k|$ of the h's.

We now relate the expansion and overweighting properties. Specifically, if h is (T, Δ)-expanding then there exists an integer $i \in \{1, ..., k\}$ so that h is $(T \cdot 2^{i-1}, \frac{\Delta}{k \cdot 2^i})$-overweighting.

Hence, at most a

$$\sum_{i=1}^{k} \frac{1}{(T \cdot 2^{i-1})^2 \cdot \frac{\Delta}{k \cdot 2^i}} < \frac{4k}{T^2 \cdot \Delta}$$

fraction of the h's can be (T, Δ)-expanding. It follows that at most $\frac{\delta}{4}$ of the h's are $(\frac{32k}{\delta^2}, \frac{\delta^3}{64k})$-expanding.

We call h *honest* if it is **not** $(\frac{32k}{\delta^2}, \frac{\delta^3}{64k})$-expanding. Hence, if h is honest and $\text{Prob}(h(X_n) \in R) \geq \frac{\delta}{2}$ then R has density at least $\frac{\delta^3}{64k}$. Concentrating on the honest h's, we now evaluate the probability that (α, h) hits B, when α is uniformly chosen. We call h *good* if $\text{Prob}((h(X_n), h) \in B) \geq \frac{\delta}{2}$. Clearly, the probability that H_n^k is good is at least $\frac{\delta}{2}$, and the probability H_n^k is both good and honest is at least $\frac{\delta}{4}$. Denote by G the set of these h's (i.e., h's which are both good and honest). Clearly, for every $h \in G$ we have $\text{Prob}((h(X_n), h) \in B) \geq \frac{\delta}{2}$ (since h is good) and $\text{Prob}((U_k, h) \in B) \geq \frac{\delta^3}{64k}$ (since h is honest). Using $\text{Prob}(H_n^k \in G) \geq \frac{\delta}{4}$, the claim follows. □

Applying Proposition 3.26

It is possible to apply Construction 3.19 to the function resulting from Construction 3.25, and the statement of Proposition 3.20 still holds with minor modifications. Specifically, Construction 3.19 is applied with $l(\cdot)$ twice the function (i.e., the $l(\cdot)$) used in Construction 3.25, and the bound in Proposition 3.20 is $3 \cdot 2^{-\frac{l(n)}{6}}$ (instead of $2^{-\frac{l(n)}{3}}$). The argument leading to Theorem 3.23, remains valid as well. Furthermore, we may even waive the requirement that $m(n)$ can be computed (since we can construct functions F_m for every possible value of $m(n)$). Finally, we note that the entire argument holds even if the definition of regular functions is relaxed as follows.

Definition 3.27 (Regular functions - revised definition): *A function $f : \{0,1\}^* \mapsto \{0,1\}^*$ is called* regular *if there exists an integer function $m' : \mathbb{N} \mapsto \mathbb{N}$ and a polynomial $q(\cdot)$ so that for all sufficiently long $x \in \{0,1\}^*$ it holds*

$$2^{m'(|x|)} \leq |\{y : f(x) = f(y)\}| \leq q(|x|) \cdot 2^{m'(|x|)}$$

When using these (relaxed) regular functions in Construction 3.25, set $m(n) \stackrel{\text{def}}{=} m'(n)$. The resulting function F will have a slightly weaker "almost" 1-1 property. Namely,

$$\text{Prob}\left(|F^{-1}F(U_n, H_n^{m(n)-l(n)})| > q(n) \cdot 2^{l(n)+1}\right) < 2^{-\frac{l(n)}{2}}$$

The application of Construction 3.19 will be modified accordingly. We get

Theorem 3.28 *If there exist regular one-way functions then pseudorandom generators exist as well.*

3.5.3 Going beyond Regular One-Way Functions

The proof of Proposition 3.26 relies heavily on the fact that the one-way function f is regular (at least in the weak sense). Alternatively, Construction 3.25 needs to be modified so that different hashing families are associated to different $x \in \{0,1\}^n$. Furthermore, the argument leading to Theorem 3.23 cannot be repeated unless it is easy to compute the cardinality of set $f^{-1}(f(x))$ given x. Note that this time we cannot construct functions F_m for every possible value of $\lceil \log_2 |f^{-1}(y)| \rceil$ since none of the functions may satisfy the statement of Proposition 3.26. Again, a new idea is needed.

A key observation is that although the value of $\log_2 |f^{-1}(f(x))|$ may vary for different $x \in \{0,1\}^n$, the value $m(n) \stackrel{\text{def}}{=} \text{Exp}(\log_2 |f^{-1}(f(U_n))|)$ is unique. Furthermore, the function $\overline{f}$ defined by

$$\overline{f}(x_1, ..., x_{n^2}) \stackrel{\text{def}}{=} f(x_1), ..., f(x_{n^2})$$

where $|x_1| = |x_{n^2}| = n$, has the property that all but a negligible fraction of the domain reside in preimage sets with logarithm of cardinality not deviating too much from the expected value. Specifically, let $\overline{m}(n^3) \stackrel{\text{def}}{=} \text{Exp}(\log_2 |\overline{f}^{-1}(\overline{f}(U_{n^3}))|)$. Clearly, $\overline{m}(n^3) = n^2 \cdot m(n)$. Using Chernoff Bound, we get

$$\text{Prob}\left(\text{abs}\left(\overline{m}(n^3) - \log_2 |\overline{f}^{-1}(\overline{f}(U_{n^3}))|\right) > n^2\right) < 2^{-n}$$

Suppose we apply Construction 3.25 to $\overline{f}$ setting $l(n^3) \stackrel{\text{def}}{=} n^2$. Denote the resulting function by $\overline{F}$. Suppose we apply Construction 3.19 to $\overline{F}$ setting this time $l(n^3) \stackrel{\text{def}}{=} 2n^2 - 1$. Using the ideas presented in the proofs of Propositions 3.20 and 3.26, one can show that if the n^3-bit to $l(n^3) + 1$-bit function used in Construction 3.19 is a hard-core of $\overline{F}$ then the resulting algorithm constitutes a pseudorandom generator. Yet, we are left with the problem of constructing a huge hard-core function, $\overline{G}$, for the function $\overline{F}$. Specifically, $|\overline{G}(x)|$ has to equal $2|x|^{\frac{2}{3}}$, for all x's. A natural idea is to define $\overline{G}$ analogously to the way $\overline{g}$ is defined in Construction 3.21. Unfortunately, we do not know how to prove the validity of this construction (when applied to $\overline{F}$), and a much more complicated construction is required. This construction does use all the above ideas in conjunction with additional ideas not presented here. The proof of validity is even more complex, and is not suitable for a book of the current nature. We thus conclude this section by merely stating the result obtained.

Theorem 3.29 *If there exist one-way functions then pseudorandom generators exist as well.*

3.6 Pseudorandom Functions

Pseudorandom generators enable to generate, exchange and share a large number of pseudorandom values at the cost of a much smaller number of random bits. Specifically, poly(n) pseudorandom bits can be generated, exchanged and shared at the cost of n (uniformly chosen bits). Since any efficient application uses only a polynomial number of random values, providing access to polynomially many pseudorandom entries seems sufficient. However, the above conclusion is too hasty, since it assumes implicitly that these entries (i.e., the addresses to be accessed) are fixed beforehand. In some natural applications, one may need to access addresses which are determined "dynamically" by the application. For example, one may want to assign random values to (poly(n) many) n-bit long strings, produced throughout the application, so that these values can be retrieved at latter time. Using pseudorandom generators the above task can be achieved at the cost of generating n random bits and storing poly(n) many values. The challenge, met in the sequel, is to achieve the above task at the cost of generating and storying only n random bits. The key to the solution is the notion of pseudorandom functions. In this section we define pseudorandom functions and show how to efficiently implement them. The implementation uses as a building block any pseudorandom generator.

3.6.1 Definitions

Loosely speaking, pseudorandom functions are functions which cannot be distinguished from truly random functions by any efficient procedure which can get the value of the function at arguments of its choice. Hence, the distinguishing procedure may query the function being examined at various points, depending possibly on previous answers obtained, and yet can not tell whether the answers were supplied by a function taken from the pseudorandom ensemble (of functions) or from the uniform ensemble (of function). Hence, to formalize the notion of pseudorandom functions we need to consider ensembles of functions. For sake of concreteness we consider in the sequel ensembles of length preserving functions. Extensions are discussed in Exercise 21.

Definition 3.30 (function ensembles): *A* function ensemble *is a sequence $F = \{F_n\}_{n\in\mathbf{N}}$ of random variables, so that the random variable F_n assumes values in the set of functions mapping n-bit long strings to n-bit long strings. The* uniform function ensemble*, denoted $H = \{H_n\}_{n\in\mathbf{N}}$, has H_n uniformly distributed over the set of functions mapping n-bit long strings to n-bit long strings.*

To formalize the notion of pseudorandom functions we use (probabilistic polynomial-time) *oracle machines*. We stress that our use of the term oracle machine is almost identical to the standard one. One deviation is that the oracle machines we consider have a length

preserving function as oracle rather than a Boolean function (as is standard in most cases in the literature). Furthermore, we assume that on input 1^n the oracle machine only makes queries of length n. These conventions are not really essential (they merely simplify the exposition a little).

Definition 3.31 (pseudorandom function ensembles): *A function ensemble, $F = \{F_n\}_{n\in\mathbb{N}}$, is called* pseudorandom *if for every probabilistic polynomial-time oracle machine M, every polynomial $p(\cdot)$ and all sufficiently large n's*

$$|\mathrm{Prob}(M^{F_n}(1^n)=1) - \mathrm{Prob}(M^{H_n}(1^n)=1)| < \frac{1}{p(n)}$$

where $H = \{H_n\}_{n\in\mathbb{N}}$ is the uniform function ensemble.

Using techniques similar to those presented in the proof of Proposition 3.3 (of Subsection 3.2.2), one can demonstrate the existence of pseudorandom function ensembles which are not statistically close to the uniform one. However, to be of practical use, we need require that the pseudorandom functions can be efficiently computed.

Definition 3.32 (efficiently computable function ensembles): *A function ensemble, $F = \{F_n\}_{n\in\mathbb{N}}$, is called* efficiently computable *if the following two conditions hold*

1. (efficient indexing): *There exists a probabilistic polynomial time algorithm, I, and a mapping from strings to functions, ϕ, so that $\phi(I(1^n))$ and F_n are identically distributed.*
   We denote by f_i the $\{0,1\}^n \mapsto \{0,1\}^n$ function assigned to i (i.e., $f_i \stackrel{\mathrm{def}}{=} \phi(i)$).
2. (efficient evaluation): *There exists a probabilistic polynomial time algorithm, V, so that $V(i,x) = f_i(x)$.*

In particular, functions in an efficiently computable function ensemble have relatively succinct representation (i.e., of polynomial rather than exponential length). It follows that efficiently computable function ensembles may have only exponentially many functions (out of the double-exponentially many possible functions).

Another point worthy of stressing is that pseudorandom functions may (if being efficiently computable) be efficiently evaluated at given points, *provided that the function description is give as well.* However, if the function (or its description) is *not* known (and it is only known that it is chosen from the pseudorandom ensemble) then the value of the function at a point cannot be approximated (even in a very liberal sense and) even if the values of the function at other points is also given.

In the rest of this book we consider only efficiently computable pseudorandom functions. Hence, in the sequel we sometimes shorthand such ensembles by calling them pseudorandom functions.

3.6.2 Construction

Using any pseudorandom generator, we construct a (efficiently computable) pseudorandom function (ensemble).

Construction 3.33 *Let G be a deterministic algorithm expanding inputs of length n into strings of length $2n$. We denote by $G_0(s)$ the $|s|$-bit long prefix of $G(s)$, and by $G_1(s)$ the $|s|$-bit long suffix of $G(s)$ (i.e., $G(s) = G_0(s)G_1(s)$). For every $s \in \{0,1\}^n$, we define a function $f_s : \{0,1\}^n \mapsto \{0,1\}^n$ so that for every $\sigma_1, ..., \sigma_n \in \{0,1\}$*

$$f_s(\sigma_1\sigma_2\cdots\sigma_n) \stackrel{\text{def}}{=} G_{\sigma_n}(\cdots(G_{\sigma_2}(G_{\sigma_1}(s)))\cdots)$$

Let F_n be a random variable defined by uniformly selecting $s \in \{0,1\}^n$ and setting $F_n = f_s$. Finally, let $F = \{F_n\}_{n\in\mathbf{N}}$ be our function ensemble.

Pictorially, the function f_s is defined by n-step walks down a full binary tree of depth n having labels on the vertices. The root of the tree, hereafter referred to as the level 0 vertex of the tree, is labelled by the string s. If an internal node is labelled r then its left child is labelled $G_0(r)$ whereas its right child is labelled $G_1(r)$. The value of $f_s(x)$ is the string residing in the leaf reachable from the root by a path corresponding to string x, when the root is labelled by s. The random variable F_n is assigned labelled trees corresponding to all possible 2^n labellings of the root, with uniform probability distribution.

A function, operating on n-bit strings, in the ensemble constructed above can be specified by n bits. Hence, selecting, exchanging and storing such a function can be implemented at the cost of selecting, exchanging and storing a single n-bit string.

Theorem 3.34 *Let G and F be as in Construction 3.33, and suppose that G is a pseudorandom generator. Then F is an efficiently computable ensemble of pseudorandom functions.*

Proof: Clearly, the ensemble F is efficiently computable. To prove that F is pseudorandom we use the hybrid technique. The k^{th} hybrid will be assigned functions which result by uniformly selecting labels for the vertices of the k^{th} (highest) level of the tree and computing the labels of lower levels as in Construction 3.33. The 0-hybrid will correspond to the random variable F_n (since a uniformly chosen label is assigned to the root), whereas the n-hybrid will correspond to the uniform random variable H_n (since a uniformly chosen label is assigned to each leaf). It will be shown that an efficient oracle machine distinguishing neighbouring hybrids can be transformed into an algorithm that distinguishes polynomially many samples of $G(U_n)$ from polynomially many samples of U_{2n}. Using Theorem 3.6 (of Subsection 3.2.3), we derive a contradiction to the hypothesis (that G is a pseudorandom generator). Details follows.

For every k, $0 \leq k \leq n$, we define a hybrid distribution H_n^k (assigned as values functions $f : \{0,1\}^n \mapsto \{0,1\}^n$) as follows. For every $s_1, s_2, ..., s_{2^k} \in \{0,1\}^n$, we define a function $f_{s_1,...,s_{2^k}} : \{0,1\}^n \mapsto \{0,1\}^n$ so that

$$f_{s_1,...,s_{2^k}}(\sigma_1\sigma_2\cdots\sigma_n) \stackrel{\text{def}}{=} G_{\sigma_n}(\cdots(G_{\sigma_{k+2}}(G_{\sigma_{k+1}}(s_{\text{idx}(\sigma_k\cdots\sigma_1)}))\cdots)$$

where $\text{idx}(\alpha)$ is index of α in the standard lexicographic order of strings of length $|\alpha|$. (In the sequel we take the liberty of associating the integer $\text{idx}(\alpha)$ with the string α.) Namely, $f_{s_{0^k},...,s_{1^k}}(x)$ is computed by first using the k-bit long prefix of x to determine one of the s_j's, and next using the $(n-k)$-bit long suffix of x to determine which of the functions G_0 and G_1 to apply at each remaining stage. The random variable H_n^k is uniformly distributed over the above $(2^n)^{2^k}$ possible functions. Namely,

$$H_n^k \stackrel{\text{def}}{=} f_{U_n^{(1)},...,U_n^{(2^k)}}$$

where $U_n^{(j)}$'s are independent random variables each uniformly distributed over $\{0,1\}^n$.

At this point it is clear that H_n^0 is identical to F_n, whereas H_n^n is identical to H_n. Again, as usual in the hybrid technique, ability to distinguish the extreme hybrids yields ability to distinguish a pair of neighbouring hybrids. This ability is further transformed (as sketched above) so that contradiction to the pseudorandomness of G is reached. Further details follow.

We assume, in contradiction to the theorem, that the function ensemble F is not pseudorandom. It follows that there exists a probabilistic polynomial-time oracle machine, M, and a polynomial $p(\cdot)$ so that for infinitely many n's

$$\Delta(n) \stackrel{\text{def}}{=} |\text{Prob}(M^{F_n}(1^n)=1) - \text{Prob}(M^{H_n}(1^n)=1)| > \frac{1}{p(n)}$$

Let $t(\cdot)$ be a polynomial bounding the running time of $M(1^n)$ (such a polynomial exists since M is polynomial-time). It follows that, on input 1^n, the oracle machine M makes at most $t(n)$ queries (since the number of queries is clearly bounded by the running time). Using the machine M, we construct an algorithm D that distinguishes the $t(\cdot)$-product of the ensemble $\{G(U_n)\}_{n\in\mathbf{N}}$ from the $t(\cdot)$-product of the ensemble $\{U_{2n}\}_{n\in\mathbf{N}}$ as follows.

On input $\alpha_1, ..., \alpha_t \in \{0,1\}^{2n}$ (with $t = t(n)$), algorithm D proceeds as follows. First, D selects uniformly $k \in \{0, 1, ..., n-1\}$. This random choice, hereafter called the *checkpoint*, and is the only random choice made by D itself. Next, algorithm D invokes the oracle machine M (on input 1^n) and answers M's queries as follows. The first query of machine M, denoted q_1, is answered by

$$G_{\sigma_n}(\cdots(G_{\sigma_{k+2}}(P_{\sigma_{k+1}}(\alpha_1)))\cdots)$$

where $q_1 = \sigma_1 \cdots \sigma_n$, and $P_0(\alpha)$ denotes the n-bit prefix of α (and $P_1(\alpha)$ denotes the n-bit suffix of α). In addition, algorithm D records this query (i.e., q_1). Subsequent queries are answered by first checking if their k-bit long prefix equals the k-bit long prefix of a previous query. In case the k-bit long prefix of the current query, denoted q_i, is different from the k-bit long prefixes of all previous queries, we associate this prefix a new input string (i.e., α_i). Namely, we answer query q_i by

$$G_{\sigma_n}(\cdots(G_{\sigma_{k+2}}(P_{\sigma_{k+1}}(\alpha_i)))\cdots)$$

where $q_i = \sigma_1 \cdots \sigma_n$. In addition, algorithm D records the current query (i.e., q_i). The other possibility is that the k-bit long prefix of the i^{th} query equals the k-bit long prefix of some previous query. Let j be the smallest integer so that the k-bit long prefix of the i^{th} query equals the k-bit long prefix of the j^{th} query (by hypothesis $j < i$). Then, we record the current query (i.e., q_i) but answer it using the string associated with query q_j. Namely, we answer query q_i by

$$G_{\sigma_n}(\cdots(G_{\sigma_{k+2}}(P_{\sigma_{k+1}}(\alpha_j)))\cdots)$$

where $q_i = \sigma_1 \cdots \sigma_n$. Finally, when machine M halts, algorithm D halts as well and outputs the same output as M.

Pictorially, algorithm D answers the first query by first placing the two halves of α_1 in the corresponding children of the tree-vertex reached by following the path from the root corresponding to $\sigma_1 \cdots \sigma_k$. The labels of all vertices in the subtree corresponding to $\sigma_1 \cdots \sigma_k$ are determined by the labels of these two children (as in the construction of F). Subsequent queries are answered by following the corresponding paths from the root. In case the path does not pass through a $(k+1)$-level vertex which has already a label, we assign this vertex and its sibling a new string (taken from the input). For sake of simplicity, in case the path of the i^{th} query requires a new string we use the i^{th} input string (rather than the first input string not used so far). In case the path of a new query passes through a $(k+1)$-level vertex which has been labelled already, we use this label to compute the labels of subsequent vertices along this path (and in particular the label of the leaf). We stress that the algorithm *does not* necessarily compute the labels of all vertices in a subtree corresponding to $\sigma_1 \cdots \sigma_k$ (although these labels are determined by the label of the vertex corresponding to $\sigma_1 \cdots \sigma_k$), but rather computes only the labels of vertices along the paths corresponding to the queries.

Clearly, algorithm D can be implemented in polynomial-time. It is left to evaluate its performance. The key observation is that when the inputs are taken from the $t(n)$-product of $G(U_n)$ and algorithm D chooses k as the checkpoint then M behaves exactly as on the k^{th} hybrid. Likewise, when the inputs are taken from the $t(n)$-product of U_{2n} and algorithm D chooses k as the checkpoint then M behaves exactly as on the $k+1^{\text{st}}$ hybrid. Namely,

Claim 3.34.1: Let n be an integer and $t \stackrel{\text{def}}{=} t(n)$. Let K be a random variable describing the random choice of checkpoint by algorithm D (on input a t-long sequence of $2n$-bit long

strings). Then for every $k \in \{0,1,...,n-1\}$

$$\begin{aligned} \text{Prob}\left(D(G(U_n^{(1)}),...,G(U_n^{(t)}))=1 \mid K=k\right) &= \text{Prob}\left(M^{H_n^k}(1^n)=1\right) \\ \text{Prob}\left(D(U_{2n}^{(1)},...,U_{2n}^{(t)})=1 \mid K=k\right) &= \text{Prob}\left(M^{H_n^{k+1}}(1^n)=1\right) \end{aligned}$$

where the $U_n^{(i)}$'s and $U_{2n}^{(j)}$'s are independent random variables uniformly distributed over $\{0,1\}^n$ and $\{0,1\}^{2n}$, respectively.

The above claim is quite obvious, yet a rigorous proof is more complex than one realizes at first glance. The reason being that M's queries may depend on previous answers it gets, and hence the correspondence between the inputs of D and possible values assigned to the hybrids is less obvious than it seems. To illustrate the difficulty consider a n-bit string which is selected by a pair of interactive processes, which proceed in n iterations. At each iteration the first party chooses a new location, based on the entire history of the interaction, and the second process sets the value of this bit by flipping an unbiased coin. It is intuitively clear that the resulting string is uniformly distributed, and the same holds if the second party sets the value of the chosen locations using the outcome of a coin flipped beforehand. In our setting the situation is slightly more involved. The process of determining the string is terminated after $k < n$ iterations and statements are made of the partially determined string. Consequently, the situation is slightly confusing and we feel that a detailed argument is required.

Proof of Claim 3.34.1: We start by sketching a proof of the claim for the extremely simple case in which M's queries are the first t strings (of length n) in lexicographic order. Let us further assume, for simplicity, that on input $\alpha_1,...,\alpha_t$, algorithm D happens to choose checkpoint k so that $t = 2^{k+1}$. In this case the oracle machine M is invoked on input 1^n and access to the function $f_{s_1,...,s_{2^{k+1}}}$, where $s_{2j-1+\sigma} = P_\sigma(\alpha_j)$ for every $j \leq 2^k$ and $\sigma \in \{0,1\}$. Thus, if the inputs to D are uniformly selected in $\{0,1\}^{2n}$ then M is invoked with access to the $k+1^{\text{st}}$ hybrid random variable (since in this case the s_j's are independent and uniformly distributed in $\{0,1\}^n$). On the other hand, if the inputs to D are distributed as $G(U_n)$ then M is invoked with access to the k^{th} hybrid random variable (since in this case $f_{s_1,...,s_{2^{k+1}}} = f_{r_1,...,r_{2^k}}$ where the r_j's are seeds corresponding to the α_j's).

For the general case we consider an alternative way of defining the random variable H_n^m, for every $0 \leq m \leq n$. This alternative way is somewhat similar to the way in which D answers the queries of the oracle machine M. (We use the symbol m instead of k since m *does not* necessarily equal the checkpoint, denoted k, chosen by algorithm D.) This way of defining H_n^m consists of the interleaving of two random processes, which together first select at random a function $g : \{0,1\}^m \mapsto \{0,1\}^n$, that is later used to determine a function $f : \{0,1\}^n \mapsto \{0,1\}^n$. The first random process, denoted ρ, is an arbitrary process ("given to us from the outside"), which specifies points in the domain of g. (The process

ρ corresponds to the queries of M, whereas the second process corresponds to the way A answers these queries.) The second process, denoted ψ, assigns uniformly selected n-bit long strings to every new point specified by ρ, thus defining the value of g on this point. We stress that in case ρ specifies an old point (i.e., a point for which g is already defined) then the second process does nothing (i.e., the value of g at this point is left unchanged). The process ρ may depend on the history of the two processes, and in particular on the values chosen for the previous points. When ρ terminates the second process (i.e., ψ) selects random values for the remaining undefined points (in case such exist). We stress that the second process (i.e., ψ) is fixed for all possible choices of a ("first") process ρ. The rest of this paragraph gives a detailed description of the interleaving of the two random processes (*and may be skipped*). We consider a randomized process ρ mapping sequences of n-bit strings (representing the history) to single m-bit strings. We stress that ρ is *not* necessarily memoryless (and hence may "remember" its previous random choices). Namely, for every fixed sequence $v_1, ..., v_i \in \{0,1\}^n$, the random variable $\rho(v_1, ..., v_i)$ is (arbitrarily) distributed over $\{0,1\}^m \cup \{\perp\}$ where $\perp$ is a special symbol denoting termination. A "random" function $g : \{0,1\}^m \mapsto \{0,1\}^n$ is defined by iterating the process ρ with the random process ψ defined below. Process ψ starts with g which is undefined on every point in its domain. At the i^{th} iteration ψ lets $p_i \stackrel{\text{def}}{=} \rho(v_1, ..., v_{i-1})$ and, assuming $p_i \neq \perp$, sets $v_i \stackrel{\text{def}}{=} v_j$ if $p_i = p_j$ for some $j < i$ and lets v_i be uniformly distributed in $\{0,1\}^n$ otherwise. In the latter case (i.e., p_i is new and hence g is not yet defined on p_i), ψ sets $g(p_i) \stackrel{\text{def}}{=} v_i$ (in fact $g(p_i) = g(p_j) = v_j = v_i$ also in case $p_i = p_j$ for some $j < i$). When ρ terminates, i.e., $\rho(v_1, ..., v_T) = \perp$ for some T, ψ completes the function g (if necessary) by choosing independently and uniformly in $\{0,1\}^n$ values for the points at which g is undefined yet. (Alternatively, we may augment the process ρ so that it terminates only after specifying all possible m-bit strings.)

Once a function g is totally defined, we define a function $f^g : \{0,1\}^n \mapsto \{0,1\}^n$ by

$$f^g(\sigma_1\sigma_2\cdots\sigma_n) \stackrel{\text{def}}{=} G_{\sigma_n}(\cdots(G_{\sigma_{k+2}}(G_{\sigma_{k+1}}(g(\sigma_k\cdots\sigma_1)))\cdots)$$

The reader can easily verify that f^g equals $f_{g(0^m),...,g(1^m)}$ (as defined in the hybrid construction above). Also, one can easily verify that the above random process (i.e., the interleaving of ψ with any ρ) yields a function g that is uniformly distributed over the set of all possible functions mapping m-bit strings to n-bit strings. It follows that the above described random process yields a result (i.e., a function) that is distributed identically to the random variable H_n^m.

Suppose now that the checkpoint chosen by D equals k and that D's inputs are independently and uniformly selected in $\{0,1\}^{2n}$. In this case the way in which D answers the M's queries can be viewed as placing independently and uniformly selected n-bit strings as the labels of the $(k+1)$-level vertices. It follows that the way in which D answers M's queries corresponds to the above described process with $m = k+1$ (with M playing the role of ρ and A playing the role of ψ). Hence, in this case M is invoked with access to the $k+1^{\text{st}}$ hybrid random variable.

Suppose, on the other hand, that the checkpoint chosen by D equals k and that D's inputs are independently selected so that each is distributed identically to $G(U_n)$. In this case the way in which D answers the M's queries can be viewed as placing independently and uniformly selected n-bit strings as the labels of the k-level vertices. It follows that the way in which D answers the M's queries corresponds to the above described process with $m = k$. Hence, in this case M is invoked with access to the k^{th} hybrid random variable. □

Using Claim 3.34.1, it follows that

$$|\text{Prob}\left(D(G(U_n^{(1)}),...,G(U_n^{(t)}))=1\right) - \text{Prob}\left(D(U_{2n}^{(1)},...,U_{2n}^{(t)})=1\right)| = \frac{\Delta(n)}{n}$$

which, by the contradiction hypothesis is greater than $\frac{1}{n \cdot p(n)}$, for infinitely many n's. Using Theorem 3.6, we derive a contradiction to the hypothesis (of the current theorem) that G is a pseudorandom generator, and the current theorem follows. ■

3.7 * Pseudorandom Permutations

In this section we present definitions and constructions for pseudorandom permutations. Clearly, pseudorandom permutations (over huge domains) can be used instead of pseudorandom functions in any efficient application, yet pseudorandom permutation offer the extra advantage of having unique preimages. This extra advantage may be useful sometimes, but not always (e.g., it is not used in the rest of this book). The construction of pseudorandom permutation uses pseudorandom functions as a building block, in a manner identical to the high level structure of the DES. Hence, the proof presented in this section can be viewed as a supporting the DES's methodology of converting "randomly looking" functions into "randomly looking" permutations. (The fact that in the DES this methodology is applied to functions which are not "randomly looking" is not of our concern here.)

3.7.1 Definitions

We start with the definition of pseudorandom permutations. Loosely speaking a pseudorandom ensemble of permutations is defined analogously to a pseudorandom ensemble of functions. Namely,

Definition 3.35 (permutation ensembles): *A* permutation ensemble *is a sequence* $P = \{P_n\}_{n\in\mathbf{N}}$ *of random variables, so that the random variable* P_n *assumes values in the set of permutations mapping* n*-bit long strings to* n*-bit long strings. The* uniform permutation ensemble*, denoted* $K = \{K_n\}_{n\in\mathbf{N}}$*, has* K_n *uniformly distributed over the set of permutations mapping* n*-bit long strings to* n*-bit long strings.*

Every permutation ensemble is a function ensemble. Hence, the definition of an *efficiently computable* permutation ensemble is obvious (i.e., it is derived from the definition of an efficiently computable function ensemble). Pseudorandom permutations are defined as computationally indistinguishable from the uniform *permutation* ensemble.

Definition 3.36 (pseudorandom permutation ensembles): *A permutation ensemble, $P = \{P_n\}_{n\in\mathbb{N}}$, is called* pseudorandom *if for every probabilistic polynomial-time oracle machine M, every polynomial $p(\cdot)$ and all sufficiently large n's*

$$|\mathrm{Prob}(M^{P_n}(1^n)=1) - \mathrm{Prob}(M^{K_n}(1^n)=1)| < \frac{1}{p(n)}$$

where $K = \{K_n\}_{n\in\mathbb{N}}$ is the uniform permutation ensemble.

The fact that P is a pseudorandom permutation ensemble rather then just being a pseudorandom function ensemble cannot be detected in poly(n)-time by an observer given oracle access to P_n. This fact steams from the observation that the uniform permutation ensemble is polynomial-time indistinguishable from the uniform function ensemble. Namely,

Proposition 3.37 *The uniform permutation ensemble (i.e., $K = \{K_n\}_{n\in\mathbb{N}}$) constitutes a pseudorandom function ensemble.*

Proof Sketch: The probability that an oracle machine detects a collision in the oracle-function, when given access to H_n, is bounded by $t^2 \cdot 2^{-n}$, where t denotes the number of queries made by the machine. Conditioned on not finding such a collision, the answers of H_n are indistinguishable from those of K_n. Finally, using the fact that a polynomial-time machine can ask at most polynomially many queries, the proposition follows. ■

Hence, using pseudorandom permutations instead of pseudorandom functions has reasons beyond the question of whether a computationally restricted observer can detect the difference. Typically, the reason is that one wants to be guaranteed of the *uniqueness* of preimages. A natural strengthening of this requirement is to require that, given the description of the permutation, *the* (unique) *preimage can be efficiently found.*

Definition 3.38 (efficiently computable and invertible permutation ensembles): *A permutation ensemble, $P = \{P_n\}_{n\in\mathbb{N}}$, is called* efficiently computable and invertible *if the following three conditions hold*

1. (efficient indexing): *There exists a probabilistic polynomial time algorithm, I, and a mapping from strings to permutation, ϕ, so that $\phi(I(1^n))$ and P_n are identically distributed.*

2. (efficient evaluation): *There exists a probabilistic polynomial time algorithm, V, so that $V(i,x) = f_i(x)$, where (as before) $f_i \stackrel{\text{def}}{=} \phi(i)$.*

3. (efficient inversion): *There exists a probabilistic polynomial time algorithm, N, so that $N(i,x) = f_i^{-1}(x)$ (i.e., $f_i(N(i,x)) = x$).*

Items (1) and (2) are guaranteed by the definition of an efficiently computable permutation ensemble. The additional requirement is stated in item (3). In some settings it makes sense to augment also the definition of a pseudorandom ensemble by requiring that the ensemble cannot be distinguished from the uniform one even when the observer gets access to two oracles: one for the permutation and the other for its inverse.

Definition 3.39 (strong pseudorandom permutations): *A permutation ensemble, $P = \{P_n\}_{n\in\mathbb{N}}$, is called* strongly pseudorandom *if for every probabilistic polynomial-time oracle machine M, every polynomial $p(\cdot)$ and all sufficiently large n's*

$$|\text{Prob}(M^{P_n,P_n^{-1}}(1^n)=1) - \text{Prob}(M^{K_n,K_n^{-1}}(1^n)=1)| < \frac{1}{p(n)}$$

where $M^{f,g}$ can ask queries to both of its oracles (e.g., query $(1,q)$ is answered by $f(q)$, whereas query $(2,q)$ is answered by $g(q)$).

3.7.2 Construction

The construction of pseudorandom permutation uses pseudorandom functions as a building block, in a manner identical to the high level structure of the DES. Namely,

Construction 3.40 *Let $f:\{0,1\}^n \mapsto \{0,1\}^n$. For every $x,y \in \{0,1\}^n$, we define*

$$\text{DES}_f(x,y) \stackrel{\text{def}}{=} (y, x \oplus f(y))$$

where $x \oplus y$ denotes the bit-by-bit exclusive-or of the binary strings x and y. Likewise, for $f_1,...,f_t:\{0,1\}^n \mapsto \{0,1\}^n$, we define

$$\text{DES}_{f_t,...,f_1}(x,y) \stackrel{\text{def}}{=} \text{DES}_{f_t,...,f_2}(\text{DES}_{f_1}(x,y))$$

For every function ensemble $F = \{F_n\}_{n\in\mathbb{N}}$, and every function $t:\mathbb{N}\mapsto\mathbb{N}$, we define the function ensemble $\{\text{DES}_{F_n}^{t(n)}\}_{n\in\mathbb{N}}$ by letting $\text{DES}_{F_n}^{t(n)} \stackrel{\text{def}}{=} \text{DES}_{F_n^{(t)},...,F_n^{(1)}}$, where $t = t(n)$ and the $F_n^{(i)}$'s are independent copies of the random variable F_n.

Theorem 3.41 *Let F_n, $t(\cdot)$, and $\mathrm{DES}^{t(n)}_{F_n}$ be as in Construction 3.40 (above). Then, for every polynomial-time computable function $t(\cdot)$, the ensemble $\{\mathrm{DES}^{t(n)}_{F_n}\}_{n\in\mathbb{N}}$ is an efficiently computable and invertible permutation ensemble. Furthermore, if $F=\{F_n\}_{n\in\mathbb{N}}$ is a pseudorandom function ensemble then the ensemble $\{\mathrm{DES}^3_{F_n}\}_{n\in\mathbb{N}}$ is pseudorandom, and the ensemble $\{\mathrm{DES}^4_{F_n}\}_{n\in\mathbb{N}}$ is strongly pseudorandom.*

Clearly, the ensemble $\{\mathrm{DES}^{t(n)}_{F_n}\}_{n\in\mathbb{N}}$ is efficiently computable. The fact that it is a permutation ensemble, and furthermore one with efficient inverting algorithm, follows from the observation that for every $x,y\in\{0,1\}^n$

$$\begin{aligned}\mathrm{DES}_{f,\mathrm{zero}}(\mathrm{DES}_f(x,y)) &= \mathrm{DES}_{f,\mathrm{zero}}(y, x\oplus f(y))\\ &= \mathrm{DES}_f(x\oplus f(y), x)\\ &= (y, (x\oplus f(y))\oplus f(y))\\ &= (x,y)\end{aligned}$$

where $\mathrm{zero}(z)\stackrel{\mathrm{def}}{=}0^{|z|}$ for all $z\in\{0,1\}^n$.

To prove the pseudorandomness of $\{\mathrm{DES}^3_{F_n}\}_{n\in\mathbb{N}}$ (resp., strong pseudorandomness of $\{\mathrm{DES}^4_{F_n}\}_{n\in\mathbb{N}}$) it suffices to prove the pseudorandomness of $\{\mathrm{DES}^3_{H_n}\}_{n\in\mathbb{N}}$ (resp., strong pseudorandomness of $\{\mathrm{DES}^4_{H_n}\}_{n\in\mathbb{N}}$). The reason being that if, say, $\{\mathrm{DES}^3_{H_n}\}_{n\in\mathbb{N}}$ is pseudorandom while $\{\mathrm{DES}^3_{F_n}\}_{n\in\mathbb{N}}$ is not, then one can derive a contradiction to the pseudorandomness of the function ensemble F (i.e., a hybrid argument is used to bridge between the three copies of H_n and the three copies of F_n). Hence, Theorem 3.41 follows from

Proposition 3.42 *$\{\mathrm{DES}^3_{H_n}\}_{n\in\mathbb{N}}$ is pseudorandom, whereas $\{\mathrm{DES}^4_{H_n}\}_{n\in\mathbb{N}}$ is strongly pseudorandom.*

Proof Sketch: We start by proving that $\{\mathrm{DES}^3_{H_n}\}_{n\in\mathbb{N}}$ is pseudorandom. Let $P_{2n}\stackrel{\mathrm{def}}{=}\{\mathrm{DES}^3_{H_n}\}_{n\in\mathbb{N}}$, and K_{2n} be the random variable uniformly distributed over all possible permutation acting on $\{0,1\}^{2n}$. We prove that for every oracle machine, M, that, on input 1^n, asks at most m queries, it holds that

$$|\mathrm{Prob}(M^{P_{2n}}(1^n)=1)-\mathrm{Prob}(M^{K_{2n}}(1^n)=1)|\leq\frac{2m^2}{2^n}$$

Let $q_i=(L^0_i,R^0_i)$, with $|L^0_i|=|R^0_i|=n$, denote the random variable representing the i^{th} query of M when given access to oracle P_{2n}. Recall that $P_{2n}=\mathrm{DES}_{H^{(3)}_n,H^{(2)}_n,H^{(1)}_n}$, where the $H^{(j)}_n$'s are three independent random variables each uniformly distributed over the functions acting on $\{0,1\}^n$. Let $R^{k+1}_i\stackrel{\mathrm{def}}{=}L^k_i\oplus H^{(k+1)}_n(R^k_i)$ and $L^{k+1}_i\stackrel{\mathrm{def}}{=}R^k_i$, for $k=0,1,2$. We assume,

without loss of generality, that M never asks the same query twice. We define the following a random variable ζ_m representing the event "there exists $i<j\leq m$ and $k\in\{1,2\}$ so that $R_i^k = R_j^k$" (namely, "on input 1^n and access to oracle P_{2n} two of the m first queries of M satisfy the relation $R_i^k = R_j^k$). Using induction on m, the reader can prove concurrently the following two claims (see guidelines below).

Claim 3.42.1: Given $\neg\zeta_m$, we have the R_i^3's uniformly distributed over $\{0,1\}^n$ and the L_i^3's uniformly distributed over the n-bit strings not assigned to previous L_j^3's. Namely, for every $\alpha_1,...,\alpha_m\in\{0,1\}^n$

$$\text{Prob}\left(\wedge_{i=1}^m(R_i^3=\alpha_i)\,|\,\neg\zeta_m\right) = \left(\frac{1}{2^n}\right)^m$$

whereas, for every *distinct* $\beta_1,...,\beta_m\in\{0,1\}^n$

$$\text{Prob}\left(\wedge_{i=1}^m(L_i^3=\beta_i)\,|\,\neg\zeta_m\right) = \prod_{i=1}^m \frac{1}{2^n-i+1}$$

Claim 3.42.2:

$$\text{Prob}\left(\zeta_{m+1}\,|\,\neg\zeta_m\right) \leq \frac{2m}{2^n}$$

Proof Idea: The proof of Claim 3.42.1 follows by observing that the R_i^3's are determined by applying the random function $H_n^{(3)}$ to different arguments (i.e., the R_i^2's), whereas the $L_i^3 = R_i^2$'s are determined by applying the random function $H_n^{(2)}$ to different arguments (i.e., the R_i^1's) and conditioning that the R_i^2's are different. The proof of Claim 3.42.2 follows by considering the probability that $R_{m+1}^k = R_i^k$, for some $i\leq m$ and $k\in\{1,2\}$. Say that $R_i^0 = R_{m+1}^0$ then certainly (by recalling $q_i\neq q_{m+1}$) we have

$$R_i^1 = L_i^0 \oplus H_n^{(1)}(R_i^0) = L_i^0 \oplus H_n^{(1)}(R_j^0) \neq L_{m+1}^0 \oplus H_n^{(1)}(R_{m+1}^0) = R_{m+1}^1$$

On the other hand, say that $R_i^0\neq R_{m+1}^0$ then

$$\text{Prob}\left(R_i^1 = R_{m+1}^1\right) = \text{Prob}\left(H_n^{(1)}(R_i^0)\oplus H_n^{(1)}(R_{m+1}^0) = L_i^0\oplus L_{m+1}^0\right) = 2^{-n}$$

Furthermore, if $R_i^1\neq R_{m+1}^1$ then

$$\text{Prob}\left(R_i^2 = R_{m+1}^2\right) = \text{Prob}\left(H_n^{(2)}(R_i^1)\oplus H_n^{(2)}(R_{m+1}^1) = R_i^0\oplus R_{m+1}^0\right) = 2^{-n}$$

Hence, both claims follow. □

Combining the above claims, we conclude that $\text{Prob}(\zeta_m) < \frac{m^2}{2^n}$, and furthermore, given that ζ_m is false, the answers of P_{2n} have left half uniformly chosen among all n-bit strings not appearing as left halves in previous answers, whereas the right half uniformly distributed among all n-bit strings. On the other hand, the answers of K_{2n} are uniformly distributed

among all $2n$-bit strings not appearing as previous answers. Hence, the statistical difference between the distribution of answers in the two cases (i.e., answers by P_{2n} or by K_{2n}) is bounded by $\frac{2m^2}{2^n}$. The first part of the proposition follows.

The proof that $\{\mathrm{DES}^4_{H_n}\}_{n\in\mathbb{N}}$ is strongly pseudorandom is more complex, yet uses essentially the same ideas. In particular, the event corresponding to ζ_m is the disjunction of four types of events. Events of the first type are of the form $R^k_i = R^k_j$ for $k \in \{2,3\}$, where $q_i = (L^0_i, R^0_i)$ and $q_j = (L^0_j, R^0_j)$ are queries of the forward direction. Similarly, events of the second type are of the form $R^k_i = R^k_j$ for $k \in \{2,1\}$, where $q_i = (L^4_i, R^4_i)$ and $q_j = (L^4_j, R^4_j)$ are queries of the backwards direction. Events of the third type are of the form $R^k_i = R^k_j$ for $k \in \{2,3\}$, where $q_i = (L^0_i, R^0_i)$ is of the forward direction, $q_j = (L^4_j, R^4_j)$ is of the backward direction, and $j < i$. Similarly, events of the fourth type are of the form $R^k_i = R^k_j$ for $k \in \{2,1\}$, where $q_i = (L^4_i, R^4_i)$ is of the forward direction, $q_j = (L^0_j, R^0_j)$ is of the backward direction, and $j < i$. As before, one bounds the probability of event ζ_m, and bounds the statistical distance between answers by K_{2n} and answers by $\{\mathrm{DES}^4_{H_n}\}_{n\in\mathbb{N}}$ given that ζ_m is false. ■

3.8 Miscellaneous

3.8.1 Historical Notes

The notion of computational indistinguishable ensembles was first presented by Goldwasser and Micali (in the context of encryption schemes) [GM82]. In the general setting, the notion first appears in Yao's work which is also the origin of the definition of pseudorandomness [Y82]. Yao also observed that pseudorandom ensembles can be very far from uniform, yet our proof of Proposition 3.3 is taken from [GK89a].

Pseudorandom generators were introduced by Blum and Micali [BM82], who defined such generators as producing sequences which are unpredictable. Blum and Micali proved that such pseudorandom generators do exist assuming the intractability of the discrete logarithm problem. Furthermore, they presented a general paradigm, for constructing pseudorandom generators, which has been used explicitly or implicitly in all subsequent developments. Other suggestions for pseudorandom generators were made soon after by Goldwasser et. al. [GMT82] and Blum et. al. [BBS82]. Consequently, Yao proved that the existence of any one-way *permutation* implies the existence of pseudorandom generators [Y82]. Yao was the first to characterize pseudorandom generators as producing sequences which are computationally indistinguishable from uniform. He also proved that this characterization of pseudorandom generators is equivalent to the characterization of Blum and Micali [BM82].

Generalizations to Yao's result, that one-way permutations imply pseudorandom generators, were proven by Levin [L85] and by Goldreich et. al. [GKL88], culminating with

the result of Hastad et. al. [H90,ILL89] which asserts that pseudorandom generators exist if and only if one-way *functions* exist. The constructions presented in Section 3.5 follow the ideas of [GKL88] and [ILL89]. These constructions make extensive use of universal$_2$ hashing functions, which were introduced by Carter and Wegman [CW] and first used in complexity theory by Sipser [S82].

Pseudorandom functions were introduced and investigated by Goldreich et. al. [GGM84]. In particular, the construction of pseudorandom functions based on pseudorandom generators is taken from [GGM84]. Pseudorandom permutations were defined and constructed by Luby and Rackoff [LR86], and our presentation follows their work.

> **Author's Note:** *Pseudorandom functions have many applications to cryptography, some of them were to be presented in other chapters of the book (e.g., on signatures and encryption). As these chapters were not written, the reader is referred to* [GGM84b] *and* [G87b,O89].

The hybrid method originates from the work of Goldwasser and Micali [GM82]. The terminology is due to Leonid Levin.

3.8.2 Suggestion for Further Reading

Section 3.5 falls short of presenting the construction of Hastad et. al. [HILL], not to mention proving its validity. Unfortunately, the proof of this fundamental theorem, asserting that pseudorandom generators exist if one-way functions exist, is too complicated to fit in a book of the current nature. The interested reader is thus referred to the original paper of Hastad et. al. [HILL] (which combines the results in [H90,ILL89]) and to Luby's book [L94book].

Simple pseudorandom generators based on specific intractability assumptions are presented in [BM82,BBS82,ACGS84,VV84,K88]. In particular, [ACGS84] presents pseudorandom generators based on the intractability of factoring, whereas [K88] presents pseudorandom generators based on the intractability of discrete logarithm problems. In both cases, the major step is the construction of hard-core predicates for the corresponding collections of one-way permutations.

Proposition 3.3 presents a pair of ensembles which are computational indistinguishable although they are statistically far apart. One of the two ensembles is not constructible in polynomial-time. Goldreich showed that a pair of polynomial-time constructible ensembles having the above property (i.e., being both computationally indistinguishable and having a non-negligibly statistical difference) exists if and only if one-way functions exist [G90ipl].

> **Author's Note:** *G90ipl has appeared in IPL, Vol. 34, pp. 277–281.*

Readers interested in Kolmogorov complexity are referred to [WHAT?]

3.8.3 Open Problems

Although Hastad et. al. [HILL] showed how to construct pseudorandom generators given any one-way *function*, their construction is not practical. The reason being that the "quality" of the generator on seeds of length n is related to the hardness of inverting the given function on inputs of length $< \sqrt[4]{n}$. We believe that presenting an efficient transformation of arbitrary one-way functions to pseudorandom generators is one of the most important open problems of the area.

An open problem of more practical importance is to try to present even more efficient pseudorandom generators based on the intractability of specific computational problems like integer factorization. For further details see Subsection 2.7.3.

3.8.4 Exercises

Exercise 1: *computational indistinguishability is preserved by efficient algorithms*: Let $\{X_n\}_{n\in\mathbb{N}}$ and $\{Y_n\}_{n\in\mathbb{N}}$ be two ensembles that are polynomial-time indistinguishable, and let A be a probabilistic polynomial-time algorithm. Prove that the ensembles $\{A(X_n)\}_{n\in\mathbb{N}}$ and $\{A(Y_n)\}_{n\in\mathbb{N}}$ are polynomial-time indistinguishable.

Exercise 2: *statistical closeness is preserved by any function*: Let $\{X_n\}_{n\in\mathbb{N}}$ and $\{Y_n\}_{n\in\mathbb{N}}$ be two ensembles that are statistically close, and let $f:\{0,1\}^* \mapsto \{0,1\}^*$ be a function. Prove that the ensembles $\{f(X_n)\}_{n\in\mathbb{N}}$ and $\{f(Y_n)\}_{n\in\mathbb{N}}$ are statistically close.

Exercise 3: Prove that for every $L \in \mathcal{BPP}$ and every pair of polynomial-time indistinguishable ensembles, $\{X_n\}_{n\in\mathbb{N}}$ and $\{Y_n\}_{n\in\mathbb{N}}$, it holds that the function

$$\Delta_L(n) \stackrel{\text{def}}{=} |\text{Prob}\,(X_n \in L) - \text{Prob}\,(Y_n \in L)\,|$$

is negligible in n.
It is tempting to think the the converse holds as well, but we don't know if it does; note that $\{X_n\}$ and $\{Y_n\}$ may be distinguished by a probabilitic algorithm, but not by a deterministic one. In such a case, which language should we define? For example, suppose that A is a probabilistic polynomial-time algorithm and let $L \stackrel{\text{def}}{=} \{x : \text{Prob}(A(x)=) \geq \frac{1}{2}\}$, then L is not necessarily in $\mathcal{BPP}$.

Exercise 4: *An equivalent formulation of statistical closeness*: In the non-computational setting both the above and its converse are true and can be easily proven. Namely, prove that two ensembles, $\{X_n\}_{n\in\mathbb{N}}$ and $\{Y_n\}_{n\in\mathbb{N}}$, are statistically close if and only if for every set $S \subseteq \{0,1\}^*$,

$$\Delta_S(n) \stackrel{\text{def}}{=} |\text{Prob}\,(X_n \in S) - \text{Prob}\,(Y_n \in S)\,|$$

is negligible in n.

Exercise 5: *statistical closeness implies computational indistinguishability*: Prove that if two ensembles are statistically close then they are polynomial-time indistinguishable. (Guideline: use the result of the previous exercise, and define for every function $f:\{0,1\}^* \mapsto \{0,1\}$ a set $S_f \stackrel{\text{def}}{=} \{x : f(x)=1\}$.)

Exercise 6: *computational indistinguishability by circuits - probabilism versus determinism*: Let $\{X_n\}_{n\in\mathbb{N}}$ and $\{Y_n\}_{n\in\mathbb{N}}$ be two ensembles, and $C \stackrel{\text{def}}{=} \{C_n\}_{n\in\mathbb{N}}$ be a family of probabilistic polynomial-size circuits. Prove that there exists a family of (deterministic) polynomial-size circuits, $D \stackrel{\text{def}}{=} \{D_n\}_{n\in\mathbb{N}}$, so that for every n

$$\Delta_D(n) \geq \Delta_C(n)$$

where

$$\begin{aligned}\Delta_D(n) &\stackrel{\text{def}}{=} |\text{Prob}\,(D_n(X_n))=1) - \text{Prob}\,(D_n(Y_n))=1)\,| \\ \Delta_C(n) &\stackrel{\text{def}}{=} |\text{Prob}\,(C_n(X_n))=1) - \text{Prob}\,(C_n(Y_n))=1)\,|\end{aligned}$$

Exercise 7: *computational indistinguishability by circuits - single sample versus several samples*: We say that the ensembles $X = \{X_n\}_{n\in\mathbb{N}}$ and $Y = \{Y_n\}_{n\in\mathbb{N}}$ are indistinguishable by polynomial-size circuits if for every family, $\{C_n\}_{n\in\mathbb{N}}$, of (deterministic) polynomial-size circuits, for every polynomial $p(\cdot)$ and all sufficiently large n's

$$|\text{Prob}\,(C_n(X_n))=1) - \text{Prob}\,(C_n(Y_n))=1)\,| < \frac{1}{p(n)}$$

Prove that X and Y are indistinguishable by polynomial-size circuits if and only if their $m(\cdot)$-products are indistinguishable by polynomial-size circuits, for every polynomial $m(\cdot)$.
(Guideline: X and Y need not be polynomial-time constructible! Yet, a "good choice" of $x^1, ..., x^k$ and $y^{k+2}, ..., y^m$ may be "hard-wired" into the circuit.)

Exercise 8: *On the general definition of a pseudorandom generator*: Let G be a pseudorandom generator (by Definition 3.8), and let $\{U_{l(n)}\}_{n\in\mathbb{N}}$ be polynomial-time indistinguishable from $\{G(U_n)\}_{n\in\mathbb{N}}$. Prove that the probability that $G(U_n)$ has length not equal to $l(n)$ is negligible (in n).
(Guideline: Consider an algorithm that for some polynomial $p(\cdot)$ proceeds as follows. On input 1^n and a string to be tested α, the algorithm first samples $G(U_n)$ for $p(n)$ times and records the length of the shortest string found. Next the algorithm outputs 1 if and only if α is longer than the length recorded.)

Exercise 9: Consider a modification of Construction 3.10, where $s_i\sigma_i = G_1(s_{i-1})$ is used instead of $\sigma_i s_i = G_1(s_{i-1})$. Provide a *simple* proof that the resulting algorithm is also pseudorandom.
(Guideline: don't modify the proof of Theorem 3.11, but rather modify G_1 itself.)

Exercise 10: Let G be a pseudorandom generator, and h be a polynomial-time computable permutation (over strings of the same length). Prove that G' and G'' defined by $G'(s) \stackrel{\text{def}}{=} h(G(s))$ and $G''(s) \stackrel{\text{def}}{=} G(h(s))$ are both pseudorandom generators.

Exercise 11: Let G be a pseudorandom generator, and h be a permutation (over strings of the same length) that is *not* necessarily polyonimial-time computable.

1. Is G' defined by $G'(s) \stackrel{\text{def}}{=} h(G(s))$ necessarily a pseudorandom generator?
2. Is G'' defined by $G''(s) \stackrel{\text{def}}{=} G(h(s))$ necessarily a pseudorandom generator?

(Guideline: you may assume that there exist one-way permutations.)

Exercise 12: *Alternative construction of pseudorandom generators with large expansion factor*: Let G_1 be a pseudorandom generator with expansion factor $l(n) = n + 1$, and let $p(\cdot)$ be a polynomial. Define $G(s)$ to be the result of applying G_1 iteratively $p(|s|)$ times on s (i.e., $G(s) \stackrel{\text{def}}{=} G_1^{p(|s|)}(s)$ where $G_1^0(s) \stackrel{\text{def}}{=} s$ and $G_1^{i+1} \stackrel{\text{def}}{=} G_1(G_1^i(s))$). Prove that G is a pseudorandom generator. What are the advantages of using Construction 3.10?

Exercise 13: *Sequential Pseudorandom Generator*: A oracle machine is called a *sequential observer* if its queries constitute a prefix of the natural numbers. Namely, on input 1^n, the sequential observer makes queries $1, 2, 3, ...$ Consider the following two experiments with a sequential observer having input 1^n:

1. The observer's queries are answered by independent flips of an unbiased coin.
2. The observer's queries are answered as follows. First a random seed, s, of length n is uniformly chosen. The i^{th} query is answered by the rightmost (i.e., the i^{th}) bit of $g_n^i(s)$, where g_n^i is defined as in the proof of Theorem 3.11.

Prove that a probabilistic polynomial-time observer cannot distinguish the two experiments, provided that G used in the construction is a pseudorandom generator. Namely, the difference between the probability that the observer outputs 1 in the first experiment and the probability that the observer outputs 1 in the second experiment is a negligible function (in n).

Exercise 14: *pseudorandomness implies unpredictability*: Prove that all pseudorandom ensembles are unpredictable (in polynomial-time).
(Guideline: Given an efficient predictor show how to construct an efficient distinguisher of the pseudorandom ensemble from the uniform one.)

Exercise 15: *unpredictability implies unpredictability*: Let $X = \{X_n\}_{n \in \mathbb{N}}$ be an ensemble such that there exists a function $l : \mathbb{N} \mapsto \mathbb{N}$ so that X_n ranges over string of length $l(n)$, and $l(n)$ can be computed in time poly(n). Prove that if X is unpredictable (in

polynomial-time) then it is pseudorandom.
(Guideline: Given an efficient distinguisher of X from the uniform ensemble $\{U_{l(n)}\}_{n\in\mathbb{N}}$ show how to construct an efficient predictor. The predictor randomly selects $k \in \{0,...,l(n)-1\}$ reads only the first k bits of the input, and applies D to the string resulting by augmenting the k-bit long prefix of the input with $l(n)-k$ uniformly chosen bits. If D answers 1 then the predictor outputs the first of these random bits else the predictor outputs the complementary value. Use a hybrid technique to evaluate the performance of the predictor. Extra hint: an argument analogous to that of the proof of Theorem 3.14 has to be used as well.)

Exercise 16: *Construction of Hashing Families*:

1. Consider the set S_n^m of functions mapping n-bit long strings into m-bit strings as follows. A function h in S_n^m is represented by an n-by-m binary matrix A, and an m-dimensional binary vector b. The n-dimensional binary vector x is mapped by the function h to the m-dimensional binary vector resulting by multiplying x by A and adding the vector b to the resulting vector (i.e., $h(x) = xA + b$). Prove that S_n^m so defined constitutes a hashing family (as defined in Section 3.5).
2. Repeat the above item when the n-by-m matrices are restricted to be Toeplitz matrices. An n-by-m *Toeplitz matrix*, $T = \{T_{i,j}\}$, satisfies $T_{i,j} = T_{i+1,j+1}$ for all i, j.

Note that binary n-by-m Toeplitz matrices can be represented by strings of length $n + m - 1$, where as representing arbitrary n-by-m binary matrices requires strings of length $n \cdot m$.

Exercise 17: *Another Hashing Lemma*: Let m, n, S_n^m, b, X_n and δ be as in Lemma 3.18. Prove that, for every set $S \subseteq \{0,1\}^m$, and for all but a $2^{-(b-m+\log_2|S|)}\delta^{-2}$ fraction of the h's in S_n^m, it holds that

$$\mathrm{Prob}(h(X_n)\in S) \in (1 \pm \delta) \cdot \frac{|S|}{2^m}$$

(Guideline: follow the proof of Lemma 3.18, defining $\zeta_x(h) = 1$ if $h(x) \in S$ and 0 otherwise.)

Exercise 18: *Yet another Hashing Lemma*: Let m, n, and S_n^m be as above. Let $B \subseteq \{0,1\}^n$ and $S \subseteq \{0,1\}^m$ be sets, and let $b \stackrel{\mathrm{def}}{=} \log_2|B|$ and $s \stackrel{\mathrm{def}}{=} \log_2|S|$. Prove that, for all but a $\frac{2^m}{|B|\cdot|S|} \cdot \delta^{-2}$ fraction of the h's in S_n^m, it holds that

$$|\{x \in B : h(x)\in S)\}| \in (1 \pm \delta) \cdot (|B| \cdot |S|)$$

(Guideline: Define a random variable X_n that is uniformly distributed over B.)

Exercise 19: *Failure of an alternative construction of pseudorandom functions*: Consider a construction of a function ensemble where the functions in F_n are defined as follows. For every $s \in \{0,1\}^n$, the function f_s is defined so that

$$f_s(x) \stackrel{\text{def}}{=} G_{\sigma_n}(\cdots(G_{\sigma_2}(G_{\sigma_1}(x)))\cdots)$$

where $s = \sigma_1 \cdots \sigma_n$, and G_σ is as in Construction 3.33. Namely the roles of x and s in Construction 3.33 are switched (i.e., the root is labelled x and the value of f_s on x is obtained by following the path corresponding to the index s). Prove that the resulting function ensemble is not necessarily pseudorandom (even if G is a pseudorandom generator).
(Guideline: Show, first, that if pseudorandom generators exist then there exists a pseudorandom generator G satisfying $G(0^n) = 0^{2n}$.)

Exercise 20: *Pseudorandom Generators with Direct Access*: A *direct access pseudorandom generator* is a deterministic polynomial-time algorithm, G, for which no probabilistic polynomial-time oracle machine can distinguish the following two cases:

1. New queries of the oracle machine are answered by independent flips of an unbiased coin. (Repeating the same query yields the same answer.)
2. First, a random "seed", s, of length n is uniformly chosen. Next, each query, q, is answered by $G(s,q)$.

The bit $G(s,i)$ may be thought of as the i^{th} bit in a bit sequence corresponding to the seed s, where i is represented in binary. Prove that the existence of (regular) pseudorandom generators implies the existence of pseudorandom generators with direct access. Note that modifying the current definition, so that only unary queries are allowed, yields an alternative definition of a sequential pseudorandom generator (presented in Exercise 13 above). Evaluate the advantage of direct access pseudorandom generators over sequential pseudorandom generators in settings requiring direct access only to bits of a polynomially long pseudorandom sequence.

Exercise 21: *other types of pseudorandom functions*: Define pseudorandom predicate ensembles so that the random variable F_n ranges over arbitrary Boolean predicates (i.e., functions in the range of F_n are defined on *all* strings and have the form $f : \{0,1\}^* \mapsto \{0,1\}$). Assuming the existence of pseudorandom generators, construct efficiently computable ensembles of pseudorandom Boolean functions. Same for ensembles of functions in which each function in the range of F_n operates on the set of all strings (i.e., has the form $f : \{0,1\}^* \mapsto \{0,1\}^*$).
(Guideline: Use a modification of Construction 3.33 in which the building block is a pseudorandom generator expanding strings of length n into strings of length $3n$.)

Exercise 22: *An alternative definition of pseudorandom functions*: For sake of simplicity this exercise is stated in terms of ensembles of Boolean functions as presented in

the previous exercise. We say that a Boolean function ensemble, $F = \{F_n\}_{n\in\mathbb{N}}$, is unpredictable if for every probabilistic polynomial-time oracle machine, M, for every polynomial $p(\cdot)$ and for all sufficiently large n's

$$\mathrm{Prob}(\mathrm{corr}^{F_n}(M^{F_n}(1^n))) < \frac{1}{2} + \frac{1}{p(n)}$$

where M^{F_n} assumes values of the form $(x,\sigma) \in \{0,1\}^{n+1}$ so that x is not a query appearing in the computation M^{F_n}, and $\mathrm{corr}^f(x,\sigma)$ is defined as the predicate "$f(x) = \sigma$". Intuitively, after getting the value of f on points of its choice, the machine M outputs a new point and tries to guess the value of f on this point. Assuming that $F = \{F_n\}_{n\in\mathbb{N}}$ is efficiently computable, prove that F is pseudorandom if and only if F is unpredictable.
(Guideline: A pseudorandom function ensemble is unpredictable since the uniform function ensemble is unpredictable. For the other direction use ideas analogous to those used in Exercise 14.)

Exercise 23: *Another alternative definition of pseudorandom functions*: Repeat the above exercise when modifying the definition of unpredictability so that the oracle machine gets $x \in \{0,1\}^n$ as input and after querying the function f on other points of its choice, the machine outputs a guess for $f(x)$. Namely, we require that for every probabilistic polynomial-time oracle machine, M, that *does not* query the oracle on its own input, for every polynomial $p(\cdot)$, and for all sufficiently large n's

$$\mathrm{Prob}(M^{F_n}(U_n) = F_n(U_n)) < \frac{1}{2} + \frac{1}{p(n)}$$

Exercise 24: Let F_n and $\mathrm{DES}^t_{F_n}$ be as in Construction 3.40. Prove that, regardless of the choice of the ensemble $F = \{F_n\}_{n\in\mathbb{N}}$, the ensemble $\mathrm{DES}^2_{F_n}$ is *not* pseudorandom. Similarly, prove that the ensemble $\mathrm{DES}^3_{F_n}$ is *not strongly* pseudorandom.
(Guideline: Start by showing that the ensemble $\mathrm{DES}^1_{F_n}$ is *not* pseudorandom.)

Chapter 4

Encryption Schemes

In this chapter we discuss the well-known notions of private-key and public-key encryption schemes. More importantly, we define what is meant by saying that such schemes are secure. We then turn to some basic constructions. We show that the widely used construction of a "stream cipher" yields a secure (private-key) encryption, provided that the "key sequence" is generated using a pseudorandom generator. Public-key encryption schemes are constructed based on any trapdoor one-way permutation. Finally, we discuss dynamic notions of security such as robustness against chosen ciphertext attacks and nonmalleability.

```
%Plan
\input{enc-set}%% The basic setting: private-key, public-key,...
\input{enc-sec}%% Definitions of Security (semantic/indistinguishable)
\input{enc-eqv}%% Equivalence of the two definitions
\input{enc-prg}%% Private-Key schemes based on Pseudorandom Generators
\input{enc-pk}%%% Constrictions of Public-Key Encryption Schemes
\input{enc-str}%% Stronger notions of security (chosen msg, ``malleable'')
\input{enc-misc}% As usual: History, Reading, Open, Exercises
```

Chapter 5

Digital Signatures and Message Authentication

The difference between message authentication and digital signatures is analogous to the difference between private-key and public-key encryption schemes. In this chapter we define both type of schemes and the security problem associated to them. We then present several constructions. We show how to construct message authentication schemes using pseudorandom functions, and how to construct signature schemes using one-way permutations (which do not necessarily have a trapdoor).

```
%Plan
\input{sg-def}%% Definitions of Unforgable Signatures
%.............. and Message Authentication
\input{sg-aut}%% Construction of Message Authentication
\input{sg-con1}%% Construction of Signatures by [NY]
%.............. tools: one-time signature, aut-trees, one-way hashing
\input{sg-hash}%% * Collision-free hashing:
%.............. def, construct by clawfree, applications (sign., etc.)
\input{sg-con2}%% * Alternative Construction of Signatures [EGM]
\input{sg-misc}%% As usual: History, Reading, Open, Exercises
```

Chapter 6

Zero-Knowledge Proof Systems

In this chapter we discuss zero-knowledge proof systems. Loosely speaking, such proof systems have the remarkable property of being convincing and yielding nothing (beyond the validity of the assertion). The main result presented is a method to generate zero-knowledge proof systems for every language in $\mathcal{NP}$. This method can be implemented using any bit commitment scheme, which in turn can be implemented using any pseudorandom generator. In addition, we discuss more refined aspects of the concept of zero-knowledge and their affect on the applicability of this concept.

6.1 Zero-Knowledge Proofs: Motivation

An archetypical "cryptographic" problem consists of providing mutually distrustful parties with a means of "exchanging" (predetermined) "pieces of information". The setting consists of several parties, each wishing to obtain some predetermined partial information concerning the secrets of the other parties. Yet each party wishes to reveal as little information as possible about its own secret. To clarify the issue, let us consider a specific example.

> Suppose that all users in a system keep backups of their entire file system, encrypted using their public-key encryption, in a publicly accessible storage media. Suppose that at some point, one user, called `Alice`, wishes to reveal to another user, called `Bob`, the cleartext of one of her files (which appears in one of her backups). A trivial "solution" is for `Alice` just to send the (cleartext) file to `Bob`. The problem with this "solution" is that `Bob` has no way of verifying that `Alice` really sent him a file from her public backup, rather than just sending him an arbitrary file. `Alice` can simply prove that she sends the correct file by revealing to `Bob` her private encryption key. However, doing so, will reveal to `Bob` the contents of all her files, which is certainly something that `Alice` does

not want to happen. The question is whether `Alice` can convince `Bob` that she indeed revealed the correct file without yielding any additional "knowledge".

An analogous question can be phrased formally as follows. Let f be a one-way *permutation*, and b a hard-core predicate with respect to f. Suppose that one party, A, has a string x, whereas another party, denoted B, only has $f(x)$. Furthermore, suppose that A wishes to reveal $b(x)$ to party B, without yielding any further information. The trivial "solution" is to let A send $b(x)$ to B, but, as explained above, B will have no way of verifying whether A has really sent the correct bit (and not its complement). Party A can indeed prove that it sends the correct bit (i.e., $b(x)$) by sending x as well, but revealing x to B is much more than what A had originally in mind. Again, the question is whether A can convince B that it indeed revealed the correct bit (i.e., $b(x)$) without yielding any additional "knowledge".

In general, the question is whether *it is possible to prove a statement without yielding anything beyond its validity.* Such proofs, whenever they exist, are called *zero-knowledge*, and play a central role (as we shall see in the subsequent chapter) in the construction of "cryptographic" protocols.

Loosely speaking, *zero-knowledge proofs are proofs that yield nothing* (i.e., "no knowledge") *beyond the validity of the assertion.* In the rest of this introductory section, we discuss the notion of a "proof" and a possible meaning of the phrase "yield nothing (i.e., no knowledge) beyond something".

6.1.1 The Notion of a Proof

We discuss the notion of a proof with the intention of uncovering some of its underlying aspects.

A Proof as a fixed sequence or as an interactive process

Traditionally in mathematics, a "proof" is a *fixed* sequence consisting of statements which are either self-evident or are derived from previous statements via self-evident rules. Actually, it is more accurate to substitute the phrase "self-evident" by the phrase "commonly agreed". In fact, in the formal study of proofs (i.e., logic), the commonly agreed statements are called *axioms*, whereas the commonly agreed rules are referred to as *derivation rules.* We wish to stress two properties of mathematics proofs:

1. proofs are viewed as fixed objects;
2. proofs are considered at least as fundamental as their consequence (i.e., the theorem).

However, in other areas of human activity, the notion of a "proof" has a much wider interpretation. In particular, a proof is not a fixed object but rather a process by which the validity of an assertion is established. For example, the cross-examination of a witness in court is considered a proof in law, and failure to answer a rival's claim is considered a proof in philosophical, political and sometimes even technical discussions. In addition, in real-life situations, proofs are considered secondary (in importance) to their consequence.

To summarize, in "canonical" mathematics proofs have a static nature (e.g., they are "written"), whereas in real-life situations proofs have a dynamic nature (i.e., they are established via an interaction). The dynamic interpretation of the notion of a proof is more adequate to our setting in which proofs are used as tools (i.e., subprotocols) inside "cryptographic" protocols. Furthermore, the dynamic interpretation (at least in a weak sense) is essential to the non-triviality of the notion of a zero-knowledge proof.

Prover and Verifier

The notion of a prover is implicit in all discussions of proofs, be it in mathematics or in real-life situations. Instead, the emphasis is placed on the *verification process*, or in other words on (the role of) the verifier. Both in mathematics and in real-life situations, proofs are defined in terms of the verification procedure. Typically, the verification procedure is considered to be relatively simple, and the burden is placed on the party/person supplying the proof (i.e., the prover).

The asymmetry between the complexity of the verification and the theorem-proving tasks is captured by the complexity class $\mathcal{NP}$, which can be viewed as a class of proof systems. Each language $L \in \mathcal{NP}$ has an efficient verification procedure for proofs of statements of the form "$x \in L$". Recall that each $L \in \mathcal{NP}$ is characterized by a polynomial-time recognizable relation R_L so that

$$L = \{x : \exists y \text{ s.t.} (x,y) \in R_L\}$$

and $(x,y) \in R_L$ only if $|y| \leq \text{poly}(|x|)$. Hence, the verification procedure for membership claims of the form "$x \in L$" consists of applying the (polynomial-time) algorithm for recognizing R_L, to the claim (encoded by) x and a prospective proof denoted y. Hence, any y satisfying $(x,y) \in R_L$ is considered a *proof* of membership of $x \in L$. Hence, correct statements (i.e., $x \in L$) and only them have proofs in this proof system. Note that the verification procedure is "easy" (i.e., polynomial-time), whereas coming up with proofs may be "difficult".

It is worthwhile to stress the distrustful attitude towards the prover in any proof system. If the verifier trusts the prover then no proof is needed. Hence, whenever discussing a proof system one considers a setting in which the verifier is not trusting the prover and furthermore is skeptic of anything the prover says.

Completeness and Validity

Two fundamental properties of a proof system (i.e., a verification procedure) are its *validity* and *completeness*. The validity property asserts that the verification procedure cannot be "tricked" into accepting false statements. In other words, *validity* captures the verifier ability of protecting itself from being convinced of false statements (no matter what the prover does in order to fool it). On the other hand, *completeness* captures the ability of some prover to convince the verifier of true statements (belonging to some predetermined set of true statements). Note that both properties are essential to the very notion of a proof system.

We remark here that not every set of true statements has a "reasonable" proof system in which each of these statements can be proven (while no false statement can be "proven"). This fundamental fact is given a precise meaning in results such as Gödel's Incompleteness Theorem and Turing's proof of the unsolvability of the Halting Problem. We stress that in this chapter we confine ourself to the class of sets that do have "efficient proof systems". In fact, Section 6.2 is devoted to discussing and formulating the concept of "efficient proof systems". Jumping ahead, we hint that the efficiency of a proof system will be associated with the efficiency of its verification procedure.

6.1.2 Gaining Knowledge

Recall that we have motivated zero-knowledge proofs as proofs by which the verifier gains "no knowledge" (beyond the validity of the assertion). The reader may rightfully wonder what is knowledge and what is a gain of knowledge. When discussing zero-knowledge proofs, we avoid the first question (which is quite complex), and treat the second question directly. Namely, *without* presenting a definition of knowledge, we present a generic case in which it is certainly justified to say that no knowledge is gained. Fortunately, this "conservative" approach seems to suffice as far as cryptography is concerned.

To motivate the definition of zero-knowledge consider a conversation between two parties, `Alice` and `Bob`. Assume first that this conversation is unidirectional, specifically `Alice` only talks and `Bob` only listens. Clearly, we can say that `Alice` gains no knowledge from the conversation. On the other hand, `Bob` may or may not gain knowledge from the conversation (depending on what `Alice` says). For example, if all that `Alice` says is $1+1=2$ then clearly `Bob` gains no knowledge from the conversation since he knows this fact himself. If, on the other hand, `Alice` tells `Bob` a proof of Fermat's Theorem then certainly he gained knowledge from the conversation.

To give a better flavour of the definition, we now consider a conversation between `Alice` and `Bob` in which `Bob` asks `Alice` questions about a large graph (that is known to both of them). Consider first the case in which `Bob` asks `Alice` whether the graph is Eulerian or not. Clearly, we say that `Bob` gains no knowledge from `Alice`'s answer, since he could have

determined the answer easily by himself (e.g., by using Euler's Theorem which asserts that a graph is Eulerian if and only if all its vertices have even degree). On the other hand, if `Bob` asks `Alice` whether the graph is Hamiltonian or not, and `Alice` (somehow) answers this question then we cannot say that `Bob` gained no knowledge (since we do not know of an efficient procedure by which `Bob` can determine the answer by himself, and assuming $\mathcal{P} \neq \mathcal{NP}$ no such efficient procedure exists). Hence, we say that `Bob` *gained knowledge* from the interaction if his *computational ability*, concerning the publicly known graph, *has increased* (i.e., if after the interaction he can easily compute something that he could not have efficiently computed before the interaction). On the other hand, if whatever `Bob` can efficiently compute about the graph *after interacting* with `Alice`, he can also efficiently compute *by himself* (from the graph) then we say that `Bob` *gained no knowledge* from the interaction. Hence, `Bob` gains knowledge only if he receives the result of a computation which is infeasible for `Bob`. The question of how could `Alice` conduct this infeasible computation (e.g., answer `Bob`'s question of whether the graph is Hamiltonian) has been ignored so far. Jumping ahead, we remark that `Alice` may be a mere abstraction or may be in possession of additional hints, that enables to efficiently conduct computations that are otherwise infeasible (and in particular are infeasible for `Bob` who does not have these hints). (Yet, these hints are not necessarily "information" in the information theoretic sense as they may be determined by the common input, but not efficiently computed from it.)

Knowledge vs. information

We wish to stress that *knowledge* (as discussed above) is very different from *information* (in the sense of information theory).

- *Knowledge* is related to computational difficulty, whereas *information* is not. In the above examples, there was a different between the knowledge revealed in case `Alice` answers questions of the form "is the graph Eulerian" and the case she answers questions of the form "is the graph Hamilton". From an information theoretic point of view there is no difference between the two cases (i.e., in both `Bob` gets no information).

- *Knowledge* relates mainly to publicly known objects, whereas *information* relates mainly to objects on which only partial information is publicly known. Consider the case in which `Alice` answers each question by flipping an unbiased coin and telling `Bob` the outcome. From an information theoretic point of view, `Bob` gets from `Alice` information concerning an event. However, we say that `Bob` gains no knowledge from `Alice`, since he can toss coins by himself.

6.2 Interactive Proof Systems

In this section we introduce the notion of an interactive proof system, and present a non-trivial example of such a system (specifically to claims of the form "the following two graphs are not isomorphic"). The presentation is directed towards the introduction of zero-knowledge interactive proofs. Interactive proof systems are interesting for their own sake, and have important complexity theoretic applications, that are discussed in Chapter 8.

6.2.1 Definition

The definition of an interactive proof system refers explicitly to the two computational tasks related to a proof system: "producing" a proof and verifying the validity of a proof. These tasks are performed by two different parties, called the *prover* and the *verifier*, which interact with one another. The interaction may be very simple and in particular unidirectional (i.e., the prover sends a text, called the proof, to the verifier). In general the interaction may be more complex, and may take the form of the verifier interrogating the prover.

Interaction

Interaction between two parties is defined in the natural manner. The only point worth noting is that the interaction is parameterized by a common input (given to both parties). In the context of interactive proof systems, the common input represents the statement to be proven. We first define the notion of an interactive machine, and next the notion of interaction between two such machines. The reader may skip to the next part of this subsection (titled "Conventions regarding interactive machines") with little loss (if at all).

Definition 6.1 (an interactive machine):

- *An* interactive Turing machine (ITM) *is a (deterministic) multi-tape Turing machine. The tapes consists of a read-only* input-tape, *a read-only* random-tape, *a read-and-write* work-tape, *a write-only* output-tape, *a pair of* communication-tapes, *and a read-and-write* switch-tape *consisting of a single cell initiated to contents 0. One communication-tape is read-only and the other is write-only.*

- *Each ITM is associated a single bit $\sigma \in \{0,1\}$, called its* identity. *An ITM is said to be* active, *in a configuration, if the contents of its switch-tape equals the machine's identity. Otherwise the machine is said to be* idle. *While being idle, the state of the machine, the location of its heads on the various tapes, and the contents of the writeable tapes of the ITM is not modified.*

- *The contents of the input-tape is called* input, *the contents of the random-tape is called* random-input, *and the contents of the output-tape at termination is called* output. *The contents written on the write-only communication-tape during a (time) period in which the machine is active is called the* message sent *at this period. Likewise, the contents read from the read-only communication-tape during an active period is called the* message received *(at that period). (Without loss of generality the machine movements on both communication-tapes are only in one direction, say left to right).*

The above definition, taken by itself, seems quite nonintuitive. In particular, one may say that once being idle the machine never becomes active again. One may also wonder what is the point of distinguishing the read-only communication-tape from the input-tape (and respectively distinguishing the write-only communication-tape from the output-tape). The point is that we are never going to consider a single interactive machine, but rather a pair of machines combined together so that some of their tapes coincide. Intuitively, the messages sent by an interactive machine are received by a second machine which shares its communication-tapes (so that the read-only communication-tape of one machine coincides with the write-only tape of the other machine). The active machine may become idle by changing the contents of the shared switch-tape and by doing so the other machine (having opposite identity) becomes active. The computation of such a pair of machines consists of the machines alternatingly sending messages to one another, based on their initial (common) input, their (distinct) random-inputs, and the messages each machine has received so far.

Definition 6.2 (joint computation of two ITMs):

- *Two interactive machines are said to be* linked *if they have opposite identities, their input-tapes coincide, their switch-tapes coincide, and the read-only communication-tape of one machine coincides with the write-only communication-tape of the other machine, and vice versa. We stress that the other tapes of both machines (i.e., the random-tape, the work-tape, and the output-tape) are distinct.*

- *The* joint computation *of a linked pair of ITMs, on a common input x, is a sequence of pairs. Each pair consists of the local configuration of each of the machines. In each such pair of local configurations, one machine (not necessarily the same one) is active while the other machine is idle.*

- *If one machine halts while the switch-tape still holds its identity the we say that both machines have halted.*

At this point, the reader may object to the above definition, saying that the individual machines are deprived of individual local inputs (and observing that they are given individual and unshared random-tapes). This restriction is removed in Subsection 6.2.3, and in

fact removing it is quite important (at least as far as practical purposes are concerned). Yet, for a first presentation of interactive proofs, as well as for demonstrating the power of this concept, we prefer the above simpler definition. The convention of individual random-tapes is however essential to the power of interactive proofs (see Exercise 4).

Conventions regarding interactive machines

Typically, we consider executions when the contents of the random-tape of each machine is uniformly and independently chosen (among all infinite bit sequences). The convention of having an infinite sequence of internal coin tosses should not bother the reader since during a finite computation only a finite prefix is read (and matters). The contents of each of these random-tapes can be viewed as internal coin tosses of the corresponding machine (as in the definition of ordinary probabilistic machines, presented in Chapter 1). Hence, interactive machines are in fact probabilistic.

Notation: Let A and B be a linked pair of ITMs, and suppose that all possible interactions of A and B on each common input terminate in a finite number of steps. We denote by $\langle A, B\rangle(x)$ the random variable representing the (local) output of B when interacting with machine A on common input x, when the random-input to each machine is uniformly and independently chosen.

Another important convention is to consider the time-complexity of an interactive machine as a function of its input only.

Definition 6.3 (the complexity of an interactive machine): *We say that an interactive machine A has* time complexity $t : \mathbb{N} \mapsto \mathbb{N}$ *if for every interactive machine B and every string x, it holds that when interacting with machine B, on common input x, machine A always (i.e., regardless of the contents of its random-tape and B's random-tape) halts within $t(|x|)$ steps.*

We stress that the time complexity, so defined, is independent of the contents of the messages that machine A receives. In other word, it is an upper bound which holds for all possible incoming messages. In particular, an interactive machine with time complexity $t(\cdot)$ reads, on input x, only a prefix of total length $t(|x|)$ of the messages sent to it.

Proof systems

In general, proof systems are defined in terms of the verification procedure (which may be viewed as one entity called the verifier). A "proof" to a specific claim is always considered as coming from the outside (which can be viewed as another entity called the prover). The

verification procedure itself, does not generate "proofs", but merely verifies their validity. Interactive proof systems are intended to capture whatever can be efficiently verified via interaction with the outside. In general, the interaction with the outside may be very complex and may consist of many message exchanges, as long as the total time spent by the verifier is polynomial.

In light of the association of efficient procedures with probabilistic polynomial-time algorithms, it is natural to consider probabilistic polynomial-time verifiers. Furthermore, the verifier's verdict of whether to accept or reject the claim is probabilistic, and a bounded error probability is allowed. (The error can of course be decreased to be negligible by repeating the verification procedure sufficiently many times.) Loosely speaking, we require that the prover can convince the verifier of the validity of valid statement, while nobody can fool the verifier into believing false statements. In fact, it is only required that the verifier accepts valid statements with "high" probability, whereas the probability that it accepts a false statement is "small" (regardless of the machine with which the verifier interacts). In the following definition, the verifier's output is interpreted as its decision on whether to accept or reject the common input. Output 1 is interpreted as 'accept', whereas output 0 is interpreted as 'reject'.

Definition 6.4 (interactive proof system): *A pair of interactive machines, (P,V), is called an* interactive proof system for a language *L if machine V is polynomial-time and the following two conditions hold*

- Completeness: *For every $x \in L$*

$$\mathrm{Prob}\left(\langle P,V\rangle(x)=1\right) \geq \frac{2}{3}$$

- Soundness: *For every $x \notin L$ and every interactive machine B*

$$\mathrm{Prob}\left(\langle B,V\rangle(x)=1\right) \leq \frac{1}{3}$$

Some remarks are in place. We first stress that the soundness condition refers to all potential "provers" whereas the completeness condition refers only to the prescribed prover P. Secondly, the verifier is required to be (probabilistic) polynomial-time, while no resource bounds are placed on the computing power of the prover (in either completeness or soundness conditions!). Thirdly, as in the case of $\mathcal{BPP}$, the error probability in the above definition can be made exponentially small by repeating the interaction (polynomially) many times (see below).

Every language in $\mathcal{NP}$ has an interactive proof system. Specifically, let $L \in \mathcal{NP}$ and let R_L be a witness relation associated with the language L (i.e., R_L is recognizable in

polynomial-time and L equals the set $\{x : \exists y \text{ s.t. } |y| = \text{poly}(x) \wedge (x,y) \in R_L\}$). Then, an interactive proof for the language L consists of a prover that on input $x \in L$ sends a witness y (as above), and a verifier that upon receiving y (on common input x) outputs 1 if $|y| = \text{poly}(|x|)$ and $(x,y) \in R_L$ (and 0 otherwise). Clearly, when interacting with the prescribed prover, this verifier will always accept inputs in the language. On the other hand, no matter what a cheating "prover" does, this verifier will never accept inputs not in the language. We point out that in this proof system both parties are deterministic (i.e., make no use of their random-tape). It is easy to see that only languages in $\mathcal{NP}$ have interactive proof systems in which both parties are deterministic (see Exercise 2).

In other words, $\mathcal{NP}$ can be viewed as a a class of interactive proof systems in which the interaction is unidirectional (i.e., from the prover to the verifier) and the verifier is deterministic (and never errs). In general interactive proofs, *both* restrictions are waived: the interaction is bidirectional and the verifier is probabilistic (and may err with some small probability). Both bidirectional interaction and randomization seem essential to the power of interactive proof systems (see further discussion in Chapter 8).

Definition 6.5 (the class $\mathcal{IP}$): *The class $\mathcal{IP}$ consists of all languages having interactive proof systems.*

By the above discussion $\mathcal{NP} \subseteq \mathcal{IP}$. Since languages in $\mathcal{BPP}$ can be viewed as having a verifier (that decides on membership without any interaction), it follows that $\mathcal{BPP} \cup \mathcal{NP} \subseteq \mathcal{IP}$. We remind the reader that it is not known whether $\mathcal{BPP} \subseteq \mathcal{NP}$.

We stress that the definition of the class $\mathcal{IP}$ remains invariant if one replaced the (constant) bounds in the completeness and soundness conditions by two functions $\mathsf{c}, \mathsf{s} : \mathbb{N} \mapsto \mathbb{N}$ satisfying $\mathsf{c}(n) < 1 - 2^{-\text{poly}(n)}$, $\mathsf{s}(n) > 2^{-\text{poly}(n)}$, and $\mathsf{c}(n) > \mathsf{s}(n) + \frac{1}{\text{poly}(n)}$. Namely,

Definition 6.6 (generalized interactive proof): *Let $\mathsf{c}, \mathsf{s} : \mathbb{N} \mapsto \mathbb{N}$ be functions satisfying $\mathsf{c}(n) > \mathsf{s}(n) + \frac{1}{p(n)}$, for some polynomial $p(\cdot)$. An interactive pair (P,V) is called a* (generalized) *interactive proof system for the language L, with* completeness bound $\mathsf{c}(\cdot)$ *and* soundness bound $\mathsf{s}(\cdot)$*, if*

- (modified) completeness: *For every $x \in L$*

$$\text{Prob}\left(\langle P,V\rangle(x)=1\right) \geq \mathsf{c}(|x|)$$

- (modified) soundness: *For every $x \notin L$ and every interactive machine B*

$$\text{Prob}\left(\langle B,V\rangle(x)=1\right) \leq \mathsf{s}(|x|)$$

The function $\mathsf{g}(\cdot)$, where $\mathsf{g}(n) \stackrel{\text{def}}{=} \mathsf{c}(n) - \mathsf{s}(n)$, is called the acceptance gap *of (P,V); and the function $\mathsf{e}(\cdot)$, where $\mathsf{e}(n) \stackrel{\text{def}}{=} \max\{1 - \mathsf{c}(n), \mathsf{s}(n)\}$, is called the* error probability *of (P,V).*

Proposition 6.7 *The following three conditions are equivalent*

1. $L \in \mathcal{IP}$. *Namely, there exists a interactive proof system, with completeness bound* $\frac{2}{3}$ *and soundness bound* $\frac{1}{3}$, *for the language* L;
2. L has very strong interactive proof systems: *For every polynomial* $p(\cdot)$, *there exists an interactive proof system for the language* L, *with error probability bounded above by* $2^{-p(\cdot)}$.
3. L has a very weak interactive proof: *There exists a polynomial* $p(\cdot)$, *and a generalized interactive proof system for the language* L, *with acceptance gap bounded below by* $1/p(\cdot)$. *Furthermore, completeness and soundness bounds for this system, namely the values* $\mathtt{c}(n)$ *and* $\mathtt{s}(n)$, *can be computed in time polynomial in* n.

Clearly either of the first two items imply the third one (including the requirement for efficiently computable bounds). The ability to efficiently compute completeness and soundness bounds is used in proving the opposite (non-trivial) direction. The proof is left as an exercise (i.e., Exercise 1).

6.2.2 An Example (Graph Non-Isomorphism in IP)

All examples of interactive proof systems presented so far were degenerate (e.g., the interaction, if at all, was unidirectional). We now present an example of a non-degenerate interactive proof system. Furthermore, we present an interactive proof system for a language *not known to be in* $\mathcal{BPP} \cup \mathcal{NP}$. Specifically, the language is the set of *pairs of non-isomorphic graphs*, denoted GNI.

Two graphs, $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, are called *isomorphic* if there exists a 1-1 and onto mapping, π, from the vertex set V_1 to the vertex set V_2 so that $(u, v) \in E_1$ if and only if $(\pi(v), \pi(u)) \in E_2$. The mapping π, if existing, is called an *isomorphism* between the graphs.

Construction 6.8 (Interactive proof system for Graph Non-Isomorphism):

- Common Input: *A pair of two graphs,* $G_1 = (V_1, E_1)$ *and* $G_2 = (V_2, E_2)$. *Suppose, without loss of generality, that* $V_1 = \{1, 2, ..., |V_1|\}$, *and similarly for* V_2.
- Verifier's first Step (V1): *The verifier selects at random one of the two input graphs, and sends to the prover a random isomorphic copy of this graph. Namely, the verifier selects uniformly* $\sigma \in \{1, 2\}$, *and a random permutation* π *from the set of permutations over the vertex set* V_σ. *The verifier constructs a graph with vertex set* V_σ *and edge set*

$$F \stackrel{\text{def}}{=} \{(\pi(u), \pi(v)) : (u, v) \in E_\sigma\}$$

and sends (V_σ, F) *to the prover.*

- Motivating Remark: *If the input graphs are non-isomorphic, as the prover claims, then the prover should be able to distinguish (not necessarily by an efficient algorithm) isomorphic copies of one graph from isomorphic copies of the other graph. However, if the input graphs are isomorphic then a random isomorphic copy of one graph is distributed identically to a random isomorphic copy of the other graph.*

- Prover's first Step (P1): *Upon receiving a graph, $G' = (V', E')$, from the verifier, the prover finds a $\tau \in \{1,2\}$ so that the graph G' is isomorphic to the input graph G_τ. (If both $\tau = 1,2$ satisfy the condition then τ is selected arbitrarily. In case no $\tau \in \{1,2\}$ satisfies the condition, τ is set to 0). The prover sends τ to the verifier.*

- Verifier's second Step (V2): *If the message, τ, received from the prover equals σ (chosen in Step V1) then the verifier outputs 1 (i.e., accepts the common input). Otherwise the verifier outputs 0 (i.e., rejects the common input).*

The verifier program presented above is easily implemented in probabilistic polynomial-time. We do not known of a probabilistic polynomial-time implementation of the prover's program, but this is not required. We now show that the above pair of interactive machines constitutes an interactive proof system (in the general sense) for the language GNI (Graph Non-Isomorphism).

Proposition 6.9 *The language GNI is in the class $\mathcal{IP}$. Furthermore, the programs specified in Construction 6.8 constitute a generalized interactive proof system for GNI. Namely,*

1. *If G_1 and G_2 are not isomorphic (i.e., $(G_1, G_2) \in GNI$) then the verifier always accept (when interacting with the prover).*

2. *If G_1 and G_2 are isomorphic (i.e., $(G_1, G_2) \notin GNI$) then, no matter with what machine the verifier interacts, it rejects the input with probability at least $\frac{1}{2}$.*

proof: Clearly, if G_1 and G_2 are not isomorphic then no graph can be isomorphic to both G_1 and G_2. It follows that there exists a unique τ such that the graph G' (received by the prover in Step P1) is isomorphic to the input graph G_τ. Hence, τ found by the prover in Step (P1) always equals σ chosen in Step (V1). Part (1) follows.

On the other hand, if G_1 and G_2 are isomorphic then the graph G' is isomorphic to both input graphs. Furthermore, we will show that in this case the graph G' yields no information about σ, and consequently no machine can (on input G_1, G_2 and G') set τ so that it equal σ, with probability greater than $\frac{1}{2}$. Details follow.

Let π be a permutation on the vertex set of a graph $G = (V, E)$. Then, we denote by $\pi(G)$ the graph with vertex set V and edge set $\{(\pi(u), \pi(v)) : (u, v) \in E\}$. Let ξ be a

random variable uniformly distributed over $\{1,2\}$, and Π be a random variable uniformly distributed over the permutations of the set V. We stress that these two random variable are independent. We are interested in the distribution of the random variable $\Pi(G_\xi)$. We are going to show that, although $\Pi(G_\xi)$ is determined by the random variables Π and ξ, the random variables ξ and $\Pi(G_\xi)$ are statistically independent. In fact we show

Claim 6.9.1: If the graphs G_1 and G_2 are isomorphic then for every graph G' it holds that

$$\text{Prob}\,(\xi=1|\Pi(G_\xi)=G') = \text{Prob}\,(\xi=2|\Pi(G_\xi)=G') = \frac{1}{2}$$

proof: We first claim that the sets $S_1 \stackrel{\text{def}}{=} \{\pi : \pi(G_1) = G')$ and $S_2 \stackrel{\text{def}}{=} \{\pi : \pi(G_2) = G')$ are of equal cardinality. This follows from the observation that there is a 1-1 and onto correspondence between the set S_1 and the set S_2 (the correspondence is given by the isomorphism between the graphs G_1 and G_2). Hence,

$$\begin{aligned}
\text{Prob}\,(\Pi(G_\xi)=G'|\xi=1) &= \text{Prob}\,(\Pi(G_1)=G') \\
&= \text{Prob}\,(\Pi \in S_1) \\
&= \text{Prob}\,(\Pi \in S_2) \\
&= \text{Prob}\,(\Pi(G_\xi)=G'|\xi=2)
\end{aligned}$$

Using Bayes Rule, the claim follows.□

Using Claim 6.9.1, it follows that for every pair, (G_1,G_2), of isomorphic graphs and for every randomized process, R, (possibly depending on this pair) it holds that

$$\begin{aligned}
\text{Prob}\,(R(\Pi(G_\xi))=\xi) &= \sum_{G'} \text{Prob}\,(\Pi(G_\xi))=G') \cdot \text{Prob}\,(R(G'))=\xi|\Pi(G_\xi)=G') \\
&= \sum_{G'} \text{Prob}\,(\Pi(G_\xi))=G') \\
&\quad \cdot \sum_{b\in\{1,2\}} \text{Prob}\,(R(G'))=b) \cdot \text{Prob}\,(b=\xi|\Pi(G_\xi)=G') \\
&= \sum_{G'} \text{Prob}\,(\Pi(G_\xi))=G') \cdot \text{Prob}\,(R(G'))\in\{1,2\}) \cdot \frac{1}{2} \\
&\leq \frac{1}{2}
\end{aligned}$$

with equality in case R always outputs an element in the set $\{1,2\}$. Part (2) of the proposition follows. ■

Remarks concerning Construction 6.8

In the proof system of Construction 6.8, the verifier *always* accepts inputs *in* the language (i.e., the error probability in these cases equals zero). All interactive proof systems we shall consider will share this property. In fact it can be shown that every interactive proof system can be *transformed* into an interactive proof system (for the same language) in which the verifier always accepts inputs in the language. On the other hand, as shown in Exercise 5, only languages in $\mathcal{NP}$ have interactive proof system in which the verifier *always rejects* inputs *not in* the language.

The fact that $GNI \in \mathcal{IP}$, whereas it is not known whether $GNI \in \mathcal{NP}$, is an indication to the power of interaction and randomness in the context of theorem proving. A much stronger indication is provided by the fact that every language in $\mathcal{PSPACE}$ has an interactive proof system (in fact $\mathcal{IP}$ equals $\mathcal{PSPACE}$). For further discussion see Chapter 8.

6.2.3 Augmentation to the Model

For purposes that will become more clear in the sequel we augment the basic definition of an interactive proof system by allowing each of the parties to have a private input (in addition to the common input). Loosely speaking, these inputs are used to capture additional information available to each of the parties. Specifically, when using interactive proof systems as subprotocols inside larger protocols, the private inputs are associated with the local configurations of the machines before entering the subprotocol. In particular, the private input of the prover may contain information which enables an efficient implementation of the prover's task.

Definition 6.10 (interactive proof systems - revisited):

- *An* interactive machine *is defined as in Definition 6.1, except that the machine has an additional read-only tape called* the auxiliary-input-tape. *The contents of this tape is call* auxiliary input.
- *The complexity of such an interactive machine is still measured as a function of the (common) input. Namely, the interactive machine A has* time complexity $t:\mathbb{N}\mapsto\mathbb{N}$ *if for every interactive machine B and every string x, it holds that when interacting with machine B, on common input x, machine A always (i.e., regardless of contents of its random-tape and its auxiliary-input-tape as well as the contents of B's tapes) halts within $t(|x|)$ steps.*
- *We denote by $\langle A(y), B(z)\rangle(x)$ the random variable representing the (local) output of B when interacting with machine A on common input x, when the random-input to each machine is uniformly and independently chosen, and A (resp., B) has auxiliary input y (resp., z).*

- *A pair of interactive machines, (P,V), is called an* interactive proof system for a language L *if machine V is polynomial-time and the following two conditions hold*
  - Completeness: *For every $x \in L$, there exists a string y such that for every $z \in \{0,1\}^*$*
    $$\text{Prob}\left(\langle P(y), V(z)\rangle(x) = 1\right) \geq \frac{2}{3}$$
  - Soundness: *For every $x \notin L$, every interactive machine B, and every $y, z \in \{0,1\}^*$*
    $$\text{Prob}\left(\langle B(y), V(z)\rangle(x) = 1\right) \leq \frac{1}{3}$$

We stress that when saying that an interactive machine is polynomial-time, we mean that its running-time is polynomial in the length of the common input. Consequently, it is not guaranteed that such a machine has enough time to read its entire auxiliary input.

6.3 Zero-Knowledge Proofs: Definitions

In this section we introduce the notion of a zero-knowledge interactive proof system, and present a non-trivial example of such a system (specifically to claims of the form "the following two graphs are isomorphic").

6.3.1 Perfect and Computational Zero-Knowledge

Loosely speaking, we say that an interactive proof system, (P,V), for a language L is zero-knowledge if whatever can be efficiently computed after interacting with P on input $x \in L$, can also be efficiently computed from x (without any interaction). We stress that the above holds with respect to any efficient way of interacting with P, not necessarily the way defined by the verifier program V. Actually, zero-knowledge is a property of the prescribed prover P. It captures P's robustness against attempts to gain knowledge by interacting with it. A straightforward way of capturing the informal discussion follows.

> Let (P,V) be an interactive proof system for some language L. We say that (P,V), actually P, is *perfect zero-knowledge* if for every probabilistic polynomial-time interactive machine V^* there exists an (*ordinary*) probabilistic polynomial-time algorithm M^* so that for every $x \in L$ the following two random variables are identically distributed
>
> - $\langle P, V^*\rangle(x)$ (i.e., the output of the interactive machine V^* after interacting with the interactive machine P on common input x);

- $M^*(x)$ (i.e., the output of machine M^* on input x).

Machine M^* is called a *simulator* for the interaction of V^* with P.

We stress that we require that *for every* V^* interacting with P, not merely for V, there exists a ("perfect") simulator M^*. This simulator, although not having access to the interactive machine P, is able to simulate the interaction of V^* with P. This fact is taken as evidence to the claim that V^* did not gain any knowledge from P (since the same output could have been generated without any access to P).

Note that every language in $\mathcal{BPP}$ has a perfect zero-knowledge proof system in which the prover does nothing (and the verifier checks by itself whether to accept the common input or not). To demonstrate the zero-knowledge property of this "dummy prover", one may present for every verifier V^* a simulator M^* which is essentially identical to V^* (except that the communication tapes of V^* are considered as ordinary work tapes of M^*).

Unfortunately, the above formulation of perfect zero-knowledge is slightly too strict to be useful. We relax the formulation by allowing the simulator to fail, with bounded probability, to produce an interaction.

Definition 6.11 (perfect zero-knowledge): *Let (P,V) be an interactive proof system for some language L. We say that (P,V) is* perfect zero-knowledge *if for every probabilistic polynomial-time interactive machine V^* there exists a probabilistic polynomial-time algorithm M^* so that for every $x \in L$ the following two conditions hold:*

1. *With probability at most $\frac{1}{2}$, on input x, machine M^* outputs a special symbol denoted $\perp$ (i.e., $\mathrm{Prob}(M^*(x)=\perp) \leq \frac{1}{2}$).*
2. *Let $m^*(x)$ be a random variable describing the distribution of $M^*(x)$ conditioned on $M^*(x) \neq \perp$ (i.e., $\mathrm{Prob}(m^*(x)=\alpha) = \mathrm{Prob}(M^*(x)=\alpha | M^*(x) \neq \perp)$, for every $\alpha \in \{0,1\}^*$). Then the following random variables are identically distributed*
   - *$\langle P, V^* \rangle(x)$ (i.e., the output of the interactive machine V^* after interacting with the interactive machine P on common input x);*
   - *$m^*(x)$ (i.e., the output of machine M^* on input x, conditioned on not being $\perp$);*

Machine M^* is called a perfect simulator *for the interaction of V^* with P.*

Condition 1 (above) can be replaced by a stronger condition requiring that M^* outputs the special symbol (i.e., $\perp$) only with negligible probability. For example, one can require that on input x machine M^* outputs $\perp$ with probability bounded above by $2^{-p(|x|)}$, for any polynomial $p(\cdot)$; see Exercise 6. Consequently, the statistical difference between the

random variables $\langle P,V^*\rangle(x)$ and $M^*(x)$ can be made negligible (in $|x|$); see Exercise 7. Hence, whatever the verifier efficiently computes after interacting with the prover, can be efficiently computed (up to an overwhelmingly small error) by the simulator (and hence by the verifier himself).

Following the spirit of Chapters 3 and 4, we observe that for practical purposes there is no need to be able to "perfectly simulate" the output of V^* after interacting with P. Instead, it suffices to generate a probability distribution which is computationally indistinguishable from the output of V^* after interacting with P. The relaxation is consistent with our original requirement that "whatever can be efficiently computed after interacting with P on input $x \in L$, can also be efficiently computed from x (without any interaction)". The reason being that we consider computationally indistinguishable ensembles as being the same. Before presenting the relaxed definition of general zero-knowledge, we recall the definition of computationally indistinguishable ensembles. Here we consider ensembles indexed by strings from a language, L. We say that the ensembles $\{R_x\}_{x\in L}$ and $\{S_x\}_{x\in L}$ are computationally indistinguishable if for every probabilistic polynomial-time algorithm, D, for every polynomial $p(\cdot)$ and all sufficiently long $x \in L$ it holds that

$$|\mathrm{Prob}(D(x,R_x)=1) - \mathrm{Prob}(D(x,S_x)=1)| < \frac{1}{p(|x|)}$$

Definition 6.12 (computational zero-knowledge): *Let (P,V) be an interactive proof system for some language L. We say that (P,V) is* computational zero-knowledge (or just zero-knowledge) *if for every probabilistic polynomial-time interactive machine V^* there exists a probabilistic polynomial-time algorithm M^* so that the following two ensembles are computationally indistinguishable*

- *$\{\langle P,V^*\rangle(x)\}_{x\in L}$ (i.e., the output of the interactive machine V^* after interacting with the interactive machine P on common input x);*
- *$\{M^*(x)\}_{x\in L}$ (i.e., the output of machine M^* on input x).*

Machine M^* is called a simulator *for the interaction of V^* with P.*

The reader can easily verify (see Exercise 9) that allowing the simulator to output the symbol $\perp$ (with probability bounded above by, say, $\frac{1}{2}$) and considering the conditional output distribution (as done in Definition 6.11), does not add to the power of Definition 6.12.

We stress that both definitions of zero-knowledge apply to interactive proof systems in the general sense (i.e., having any non-negligible gap in the acceptance probabilities for inputs inside and outside the language). In fact, the definitions of zero-knowledge apply to

any pair of interactive machines (actually to each interactive machine). Namely, we may say that the interactive machine A is *zero-knowledge on* L if whatever can be efficiently computed after interacting with A on common input $x \in L$, can also be efficiently computed from x itself.

An alternative formulation of zero-knowledge

An alternative formulation of zero-knowledge considers the verifier's view of the interaction with the prover, rather than only the output of the verifier after such an interaction. By the "verifier's view of the interaction" we mean the entire sequence of the local configurations of the verifier during an interaction (execution) with the prover. Clearly, it suffices to consider only the contents of the random-tape of the verifier and the sequence of messages that the verifier has received from the prover during the execution (since the entire sequence of local configurations as well as the final output are determine by these objects).

Definition 6.13 (zero-knowledge – alternative formulation): *Let (P,V), L and V^* be as in Definition 6.12. We denote by* $\mathrm{view}^P_{V^*}(x)$ *a random variable describing the contents of the random-tape of V^* and the messages V^* receives from P during a joint computation on common input x. We say that (P,V) is* zero-knowledge *if for every probabilistic polynomial-time interactive machine V^* there exists a probabilistic polynomial-time algorithm M^* so that the ensembles $\{\mathrm{view}^P_{V^*}(x)\}_{x\in L}$ and $\{M^*(x)\}_{x\in L}$ are computationally indistinguishable.*

A few remarks are in place. Definition 6.13 is obtained from Definition 6.12 by replacing $\langle P,V^*\rangle(x)$ for $\mathrm{view}^P_{V^*}(x)$. The simulator M^* used in Definition 6.13 is related, but not equal, to the simulator used in Definition 6.12 (yet, this fact is not reflected in the text of these definitions). Clearly, $V^*(x)$ can be computed in (deterministic) polynomial-time from $\mathrm{view}^P_{V^*}(x)$, for every V^*. Although the opposite direction is not always true, Definition 6.13 is equivalent to Definition 6.12 (see Exercise 10). The latter fact justifies the use of Definition 6.13, which is more convenient to work with, although it seems less natural than Definition 6.12. An alternative formulation of perfect zero-knowledge is straightforward, and clearly it is equivalent to Definition 6.11.

* Complexity classes based on Zero-Knowledge

Definition 6.14 (class of languages having zero-knowledge proofs): *We denote by $\mathcal{ZK}$* (also $\mathcal{CZK}$) *the class of languages having* (computational) *zero-knowledge interactive proof systems. Likewise, $\mathcal{PZK}$ denotes the class of languages having* perfect *zero-knowledge interactive proof systems.*

Clearly, $\mathcal{BPP} \subseteq \mathcal{PZK} \subseteq \mathcal{CZK} \subseteq \mathcal{IP}$. We believe that the first two inclusions are strict. Assuming the existence of (non-uniformly) one-way functions, the last inclusion is an equality (i.e., $\mathcal{CZK} = \mathcal{IP}$). See Proposition 6.24 and Theorems 3.29 and 6.30.

* Expected polynomial-time simulators

The formulation of perfect zero-knowledge presented in Definition 6.11 is different from the standard definition used in the literature. The standard definition requires that the simulator always outputs a legal transcript (which has to be distributed identically to the real interaction) yet it allows the simulator to run in *expected* polynomial-time rather than in strictly polynomial-time time. We stress that the expectation is taken over the coin tosses of the simulator (whereas the input to the simulator is fixed).

Definition 6.15 (perfect zero-knowledge – liberal formulation): *We say that* (P,V) *is* perfect zero-knowledge in the liberal sense *if for every* probabilistic *polynomial-time interactive machine* V^* *there exists an* expected *polynomial-time algorithm* M^* *so that for every* $x \in L$ *the random variables* $\langle P, V^* \rangle(x)$ *and* $M^*(x)$ *are identically distributed.*

We stress that by *probabilistic polynomial-time* we mean a strict bound on the running time in all possible executions, whereas by *expected polynomial-time* we allow non-polynomial-time executions but require that the running-time is "polynomial on the average". Clearly, Definition 6.11 implies Definition 6.15 – see Exercise 8. Interestingly, there exists interactive proofs which are perfect zero-knowledge with respect to the liberal definition but not known to be perfect zero-knowledge with respect to Definition 6.11. We prefer to adopt Definition 6.11, rather than Definition 6.15, because we wanted to avoid the notion of expected polynomial-time that is much more subtle than one realizes at first glance.

> *A parenthetical remark concerning the notion of average polynomial-time*: The naive interpretation of expected polynomial-time is having *average* running-time that is *bounded by a polynomial* in the input length. This definition of expected polynomial-time is unsatisfactory since it is *not closed under reductions* and is (too) *machine dependent*. Both aggravating phenomenon follow from the fact that a function may have an average (say over $\{0,1\}^n$) that is bounded by polynomial (in n) and yet squaring the function yields a function which is not bounded by a polynomial (in n). Hence, a better interpretation of expected polynomial-time is having running-time that is *bounded by a polynomial in a function which has average linear growing rate.*

Furthermore, the correspondence between average polynomial-time and efficient computations is more controversial than the more standard association of strict polynomial-time with efficient computations.

An analogous discussion applies also to computational zero-knowledge. More specifically, Definition 6.12 requires that the simulator works in polynomial-time, whereas a more liberal notion allows it to work in *expected* polynomial-time.

For sake of elegancy, it is customary to modify the definitions allowing *expected* polynomial-time simulators, by requiring that such simulators exist also for the interaction of *expected* polynomial-time verifiers with the prover.

6.3.2 An Example (Graph Isomorphism in PZK)

As mentioned above, every language in $\mathcal{BPP}$ has a trivial (i.e., degenerate) zero-knowledge proof system. We now present an example of a non-degenerate zero-knowledge proof system. Furthermore, we present a zero-knowledge proof system for a language not known to be in $\mathcal{BPP}$. Specifically, the language is the set of *pairs of isomorphic graphs*, denoted GI (see definition in Section 6.2).

Construction 6.16 (Perfect Zero-Knowledge proof for Graph Isomorphism):

- Common Input: *A pair of two graphs, $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$. Let ϕ be an isomorphism between the input graphs, namely ϕ is a 1-1 and onto mapping of the vertex set V_1 to the vertex set V_2 so that $(u, v) \in E_1$ if and only if $(\pi(v), \pi(u)) \in E_2$.*

- Prover's first Step (P1): *The prover selects a random isomorphic copy of G_2, and sends it to the verifier. Namely, the prover selects at random, with uniform probability distribution, a permutation π from the set of permutations over the vertex set V_2, and constructs a graph with vertex set V_2 and edge set*

  $$F \stackrel{\text{def}}{=} \{(\pi(u), \pi(v)) : (u, v) \in E_2\}$$

  The prover sends (V_2, F) to the verifier.

- Motivating Remark: *If the input graphs are isomorphic, as the prover claims, then the graph sent in step P1 is isomorphic to both input graphs. However, if the input graphs are* not *isomorphic then no graph can be isomorphic to both of them.*

- Verifier's first Step (V1): *Upon receiving a graph, $G' = (V', E')$, from the prover, the verifiers asks the prover to show an isomorphism between G' and one of the input graph, chosen at random by the verifier. Namely, the verifier uniformly selects $\sigma \in \{1, 2\}$, and sends it to the prover (who is supposed to answer with an isomorphism between G_σ and G').*

- Prover's second Step (P2): *If the message, σ, received from the verifier equals 2 then the prover sends π to the verifier. Otherwise (i.e., $\sigma \neq 2$), the prover sends $\pi \circ \phi$ (i.e., the composition of π on ϕ, defined as $\pi \circ \phi(v) \stackrel{\text{def}}{=} \pi(\phi(v))$) to the verifier. (Remark: the prover treats any $\sigma \neq 2$ as $\sigma = 1$.)*

- Verifier's second Step (V2): *If the message, denoted ψ, received from the prover is an isomorphism between G_σ and G' then the verifier outputs 1, otherwise it outputs 0.*

Let use denote the prover's program by P_{GI}.

The verifier program presented above is easily implemented in probabilistic polynomial-time. In case the prover is given an isomorphism between the input graphs as auxiliary input, also the prover's program can be implemented in probabilistic polynomial-time. We now show that the above pair of interactive machines constitutes a *zero-knowledge* interactive proof system (in the general sense) for the language GI (Graph Isomorphism).

Proposition 6.17 *The language GI has a perfect zero-knowledge interactive proof system. Furthermore, the programs specified in Construction 6.16 satisfy*

1. *If G_1 and G_2 are isomorphic (i.e., $(G_1, G_2) \in GI$) then the verifier always accepts (when interacting with the prover).*
2. *If G_1 and G_2 are not isomorphic (i.e., $(G_1, G_2) \notin GI$) then, no matter with what machine the verifier interacts, it rejects the input with probability at least $\frac{1}{2}$.*
3. *The above prover (i.e., P_{GI}) is perfect zero-knowledge. Namely, for every probabilistic polynomial-time interactive machine V^* there exists a probabilistic polynomial-time algorithm M^* outputting $\perp$ with probability at most $\frac{1}{2}$ so that for every $x \stackrel{\text{def}}{=} (G_1, G_2) \in GI$ the following two random variables are identically distributed*
   - $\text{view}_{V^*}^{P_{GI}}(x)$ *(i.e., the view of V^* after interacting with P_{GI}, on common input x);*
   - $m^*(x)$ *(i.e., the output of machine M^*, on input x, conditioned on not being $\perp$).*

A zero-knowledge interactive proof system for GI with error probability 2^{-k} (only in the soundness condition) can be derived by executing the above protocol, sequentially, k times. We stress that in each repetition, of the above protocol, both (the prescribed) prover and verifier use coin tosses which are independent of the coins used in the other repetitions of the protocol. For further discussion see Section 6.3.4. We remark that k parallel executions will decrease the error in the soundness condition to 2^{-k} as well, but the resulting interactive proof is not known to be zero-knowledge in case k grows faster than logarithmic in the input length. In fact, we believe that such an interactive proof is *not* zero-knowledge. For further discussion see Section 6.5.

We stress that it is not known whether $GI \in \mathcal{BPP}$. Hence, Proposition 6.17 asserts the existence of perfect zero-knowledge proofs for languages not known to be in $\mathcal{BPP}$.

proof: We first show that the above programs indeed constitute a (general) interactive proof system for GI. Clearly, if the input graphs, G_1 and G_2, are isomorphic then the graph G'

constructed in step (P1) is isomorphic to both of them. Hence, if each party follows its prescribed program then the verifier always accepts (i.e., outputs 1). Part (1) follows. On the other hand, if G_1 and G_2 are not isomorphic then no graph can be isomorphic to both G_1 and G_2. It follows that no matter how the (possibly cheating) prover constructs G' there exists $\sigma \in \{1,2\}$ so that G' and G_σ are *not* isomorphic. Hence, when the verifier follows its program, the verifier rejects (i.e., outputs 0) with probability at least $\frac{1}{2}$. Part (2) follows.

It remains to show that P_{GI} is indeed perfect zero-knowledge on GI. This is indeed the difficult part of the entire proof. It is easy to simulate the output of the verifier specified in Construction 6.16 (since its output is identically 1 on inputs in the language GI). It is also not hard to simulate the output of a verifier which follows the program specified in Construction 6.16, except that at termination it output the entire transcript of its interaction with P_{GI} – see Exercise 11. The difficult part is to simulate the output of an efficient verifier which deviates arbitrarily from the specified program.

We will use here the alternative formulation of (perfect) zero-knowledge, and show how to simulate V^*'s view of the interaction with P_{GI}, for every probabilistic polynomial-time interactive machine V^*. As mentioned above it is not hard to simulate the verifier's view of the interaction with P_{GI} in case the verifier follows the specified program. However, we need to simulate the view of the verifier in the general case (in which it uses an arbitrary polynomial-time interactive program). Following is an overview of our simulation (i.e., of our construction of a simulator, M^*, for each V^*).

The simulator M^* incorporates the code of the interactive program V^*. On input (G_1, G_2), the simulator M^* first selects at random one of the input graphs (i.e., either G_1 or G_2) and generates a random isomorphic copy, denoted G'', of this input graph. In doing so, the simulator behaves differently from P_{GI}, but the graph generated (i.e., G'') is distributed identically to the message sent in step (P1) of the interactive proof. Say that the simulator has generated G'' by randomly permuting G_1. Then, if V^* asks to see the isomorphism between G_1 and G'', the simulator can indeed answer correctly and in doing so it completes a simulation of the verifier's view of the interaction with P_{GI}. However, if V^* asks to see the isomorphism between G_2 and G'', then the simulator (which, unlike P_{GI}, does not "know" ϕ) has no way to answer correctly, and we let it halt with output $\perp$. We stress that the simulator "has no way of knowing" whether V^* will ask to see an isomorphism to G_1 or G_2. The point is that the simulator can try one of the possibilities at random and if it is lucky (which happens with probability exactly $\frac{1}{2}$) then it can output a distribution which is identical to the view of V^* when interacting with P_{GI} (on common input (G_1, G_2)). A detailed description of the simulator follows.

Simulator M^*. On input $x \stackrel{\text{def}}{=} (G_1, G_2)$, simulator M^* proceeds as follows:

1. *Setting the random-tape of* V^*: Let $q(\cdot)$ denote a polynomial bounding the running-time of V^*. The simulator M^* starts by uniformly selecting a string $r \in \{0,1\}^{q(|x|)}$, to be used as the contents of the random-tape of V^*.

2. *Simulating the prover's first step (P1):* The simulator M^* selects at random, with uniform probability distribution, a "bit" $\tau \in \{1,2\}$ and a permutation ψ from the set of permutations over the vertex set V_τ. It then constructs a graph with vertex set V_τ and edge set
$$F \stackrel{\text{def}}{=} \{(\psi(u),\psi(v)) : (u,v) \in E_\tau\}$$
Set $G'' \stackrel{\text{def}}{=} (V_\tau, F)$.

3. *Simulating the verifier's first step (V1):* The simulator M^* initiates an execution of V^* by placing x on V^*'s common-input-tape, placing r (selected in step (1) above) on V^*'s random-tape, and placing G'' (constructed in step (2) above) on V^*'s incoming message-tape. After executing a polynomial number of steps of V^*, the simulator can read the outgoing message of V^*, denoted σ. To simplify the rest of the description, we *normalize* σ by setting $\sigma = 1$ if $\sigma \neq 2$ (and leave σ unchanged if $\sigma = 2$).

4. *Simulating the prover's second step (P2):* If $\sigma = \tau$ then the simulator halts with output (x, r, G'', ψ).

5. *Failure of the simulation:* Otherwise (i.e., $\sigma \neq \tau$), the simulator halts with output $\perp$.

Using the hypothesis that V^* is polynomial-time, it follows that so is the simulator M^*. It is left to show that M^* outputs $\perp$ with probability at most $\frac{1}{2}$, and that, conditioned on not outputting $\perp$, the simulator's output is distributed as the verifier's view in a "real interaction with P_{GI}". The following claim is the key to the proof of both claims.

Claim 6.17.1: Suppose that the graphs G_1 and G_2 are isomorphic. Let ξ be a random variable uniformly distributed in $\{1,2\}$, and $\Pi(G)$ be a random variable (independent of ξ) describing the graph obtained from the graph G by randomly relabelling its nodes (cf. Claim 6.9.1). Then, for every graph G'', it holds that
$$\text{Prob}\,(\xi = 1 | \Pi(G_\xi) = G'') = \text{Prob}\,(\xi = 2 | \Pi(G_\xi) = G'')$$

Claim 6.17.1 is identical to Claim 6.9.1 (used to demonstrate that Construction 6.8 constitutes an interactive proof for GNI). As in the rest of the proof of Proposition 6.9, it follows that any random process with output in $\{1,2\}$, given $\Pi(G_\xi)$, outputs ξ with probability exactly $\frac{1}{2}$. Hence, given G'' (constructed by the simulator in step (2)), the verifier's program yields (normalized) σ so that $\sigma \neq \tau$ with probability exactly $\frac{1}{2}$. We conclude that the simulator outputs $\perp$ with probability $\frac{1}{2}$. It remains to prove that, conditioned on not outputting $\perp$, the simulator's output is identical to "V^*'s view of real interactions". Namely,

Claim 6.17.2: Let $x = (G_1, G_2) \in GI$. Then, for every string r, graph H, and permutation ψ, it holds that
$$\text{Prob}\left(\text{view}^{P_{GI}}_{V^*}(x) = (x, r, H, \psi)\right) = \text{Prob}\,(M^*(x) = (x, r, H, \psi) \,|\, M^*(x) \neq \perp)$$

proof: Let $m^*(x)$ describe $M^*(x)$ conditioned on its not being $\perp$. We first observe that both $m^*(x)$ and $\text{view}_{V^*}^{P_{GI}}(x)$ are distributed over quadruples of the form $(x,r,\cdot,\cdot)$, with uniformly distributed $r \in \{0,1\}^{q(|x|)}$. Let $\nu(x,r)$ be a random variable describing the last two elements of $\text{view}_{V^*}^{P_{GI}}(x)$ conditioned on the second element equals r. Similarly, let $\mu(x,r)$ describe the last two elements of $m^*(x)$ (conditioned on the second element equals r). Clearly, it suffices to show that $\nu(x,r)$ and $\mu(x,r)$ are identically distributed, for every x and r. Observe that once r is fixed the message sent by V^* on common input x, random-tape r, and incoming message H, is uniquely defined. Let us denote this message by $v^*(x,r,H)$. We show that both $\nu(x,r)$ and $\mu(x,r)$ are uniformly distributed over the set

$$C_{x,r} \stackrel{\text{def}}{=} \left\{(H,\psi) : H = \psi(G_{v^*(x,r,H)})\right\}$$

where $\psi(G)$ denotes the graph obtained from G by relabelling the vertices using the permutation ψ (i.e., if $G=(V,E)$ then $\psi(G)=(V,F)$ so that $(u,v)\in E$ iff $(\psi(u),\psi(v))\in F$). The proof of this statement is rather tedious and unrelated to the subjects of this book (and hence can be skipped with no damage).

The proof is slightly non-trivial because it relates (at least implicitly) to the automorphism group of the graph G_2 (i.e., the set of permutations π for which $\pi(G_2)$ is identical, not just isomorphic, to G_2). For simplicity, consider first the special case in which the automorphism group of G_2 consists of merely the identity permutation (i.e., $G_2=\pi(G_2)$ if and only if π is the identity permutation). In this case, $(H,\psi)\in C_{x,r}$ if and only if H is isomorphic to (both G_1 and) G_2 and ψ is the isomorphism between H and $G_{v^*(x,r,H)}$. Hence, $C_{x,r}$ contains exactly $|V_2|!$ pairs, each containing a different graph H as the first element. In the general case, $(H,\psi)\in C_{x,r}$ if and only if H is isomorphic to (both G_1 and) G_2 and ψ is an isomorphism between H and $G_{v^*(x,r,H)}$. We stress that $v^*(x,r,H)$ is the same in all pairs containing H. Let $\text{aut}(G_2)$ denotes the size of the automorphism group of G_2. Then, each H (isomorphic to G_2) appears in exactly $\text{aut}(G_2)$ pairs of $C_{x,r}$ and each such pair contain a different isomorphism between H and $G_{v^*(x,r,H)}$.

We first consider the random variable $\mu(x,r)$ (describing the suffix of $m^*(x)$). Recall that $\mu(x,r)$ is defined by the following two step random process. In the *first* step, one selects uniformly a pair (τ,ψ), over the set of pairs $\{1,2\}$-times-permutation, and sets $H=\psi(G_\tau)$. In the *second* step, one outputs (i.e., sets $\mu(x,r)$ to) $(\psi(G_\tau),\psi)$ if $v^*(x,r,H)=\tau$ (and ignores the (τ,ψ) pair otherwise). Hence, each graph H (isomorphic to G_2) is generated, at the first step, by exactly $\text{aut}(G_2)$ different $(1,\cdot)$-pairs (i.e., the pairs $(1,\psi)$ satisfying $H=\psi(G_1)$), and by exactly $\text{aut}(G_2)$ different $(2,\cdot)$-pairs (i.e., the pairs $(2,\psi)$ satisfying $H=\psi(G_2)$). All these $2\cdot\text{aut}(G_2)$ pairs yield the same graph H, and hence lead to the same value of $v^*(x,r,H)$. It follows that out of the $2\cdot\text{aut}(G_2)$ pairs, (τ,ψ), yielding

the graph $H = \psi(G_\tau)$, only the pairs satisfying $\tau = v^*(x, r, H)$ lead to an output. Hence, for each H (which is isomorphic to G_2), the probability that $\mu(x,r) = (H, \cdot)$ equals $\mathrm{aut}(G_2)/(|V_2|!)$. Furthermore, for each H (which is isomorphic to G_2),

$$\mathrm{Prob}\,(\mu(x,r) = (H,\psi)) = \begin{cases} \frac{1}{|V_2|!} & \text{if} \quad H = \psi(G_{v^*(x,r,H)}) \\ 0 & \text{otherwise} \end{cases}$$

Hence $\mu(x,r)$ is uniformly distributed over $C_{x,r}$.

We now consider the random variable $\nu(x,r)$ (describing the suffix of the verifier's view in a "real interaction" with the prover). Recall that $\nu(x,r)$ is defined by selecting uniformly a permutation π (over the set V_2), and setting $\nu(x,r) = (\pi(G_2), \pi)$ if $v^*(x, r, \pi(G_2)) = 2$ and $\nu(x,r) = (\pi(G_2), \pi \circ \phi)$ otherwise, where ϕ is the isomorphism between G_1 and G_2. Clearly, for each H (which is isomorphic to G_2), the probability that $\nu(x,r) = (H, \cdot)$ equals $\mathrm{aut}(G_2)/(|V_2|!)$. Furthermore, for each H (which is isomorphic to G_2),

$$\mathrm{Prob}\,(\nu(x,r) = (H,\psi)) = \begin{cases} \frac{1}{|V_2|!} & \text{if} \quad \psi = \pi \circ \phi^{2-v^*(x,r,H)} \\ 0 & \text{otherwise} \end{cases}$$

Observing that $H = \psi(G_{v^*(x,r,H)})$ if and only if $\psi = \pi \circ \phi^{2-v^*(x,r,H)}$, we conclude that $\mu(x,r)$ and $\nu(x,r)$ are identically distributed.

The claim follows. □

This completes the proof of Part (3) of the proposition. ■

6.3.3 Zero-Knowledge w.r.t. Auxiliary Inputs

The definitions of zero-knowledge presented above fall short of what is required in practical applications and consequently a minor modification should be used. We recall that these definitions guarantee that whatever can be efficiently computed after interaction with the prover on any *common input*, can be efficiently computed *from the input itself*. However, in typical applications (e.g., when an interactive proof is used as a sub-protocol inside a bigger protocol) the verifier interacting with the prover, on common input x, may have some additional a-priori information, encoded by a string z, which may assist it in its attempts to "extract knowledge" from the prover. This danger may become even more acute in the likely case in which z is related to x. (For example, consider the protocol of Construction 6.16 and the case where the verifier has a-priori information concerning an isomorphism between the input graphs.) What is typically required is that whatever can be efficiently computed from x and z after interaction with the prover on any common input x, can be efficiently computed from x and z (without any interaction with the prover). This requirement is formulated below using the augmented notion of interactive proofs presented in Definition 6.10.

Definition 6.18 (zero-knowledge – revisited): *Let (P,V) be an interactive proof for a language L (as in Definition 6.10). Denote by $P_L(x)$ the set of strings y satisfying the completeness condition with respect to $x \in L$ (i.e., for every $z \in \{0,1\}^*$ $\mathrm{Prob}(\langle P(y),V(z)\rangle(x)=1) \geq \frac{2}{3}$). We say that (P,V) is* zero-knowledge with respect to auxiliary input (auxiliary input zero-knowledge) *if for every probabilistic polynomial-time interactive machine V^* there exists a probabilistic algorithm M^*, running in time polynomial in the length of its first input, so that the following two ensembles are computationally indistinguishable (when the distinguishing gap is considered as a function of $|x|$)*

- $\{\langle P(y),V^*(z)\rangle(x)\}_{x\in L, y\in P_L(x), z\in\{0,1\}^*}$
- $\{M^*(x,z)\}_{x\in L, z\in\{0,1\}^*}$

Namely, for every probabilistic algorithm, D, with running-time polynomial in length of the first input, every polynomial $p(\cdot)$, and all sufficiently long $x \in L$, all $y \in P_L(x)$ and $z \in \{0,1\}^*$, it holds that

$$|\mathrm{Prob}(D(x,z,\langle P(y),V^*(z)\rangle(x))=1) - \mathrm{Prob}(D(x,z,M^*(x,z))=1)| < \frac{1}{p(|x|)}$$

In the above definition y represents a-priori information to the prover, whereas z represents a-priori information to the verifier. Both y and z may depend on the common input x. We stress that the local inputs (i.e., y and z) may not be known, even in part, to the counterpart. We also stress that the auxiliary input z is also given to the distinguishing algorithm (which may be thought of as an extension of the verifier).

Recall that by Definition 6.10, saying that the interactive machine V^* is probabilistic polynomial-time means that its running-time is bounded by a polynomial in the length of the common input. Hence, the verifier program, the simulator, and the distinguishing algorithm, all run in time polynomial in the length of x (and not in time polynomial in the total length of all their inputs). This convention is essential in many respects. For example, having allowed even one of these machines to run in time proportional to the length of the auxiliary input would have collapsed computational zero-knowledge to perfect zero-knowledge (e.g., by considering verifiers which run in time polynomial in the common-input yet have huge auxiliary inputs of length exponential in the common-input).

Definition 6.18 refers to computational zero-knowledge. A formulation of perfect zero-knowledge with respect to auxiliary input is straightforward. We remark that the perfect zero-knowledge proof for Graph Isomorphism, presented in Construction 6.16, is in fact perfect zero-knowledge with respect to auxiliary input. This fact follows easily by a minor augmentation to the simulator constructed in the proof of Proposition 6.17 (i.e., when invoking the verifier, the simulator should provide it with the auxiliary input which is given to the simulator). In general, a demonstration of zero-knowledge can be extended

to yield zero-knowledge with respect to auxiliary input, provided that the simulator used in the original demonstration works by invoking the verifier's program as a black box. All simulators presented in this book have this property.

*** Implicit non-uniformity in Definition 6.18**

The non-uniform nature of Definition 6.18 is captured by the fact that the simulator gets an auxiliary input. It is true that this auxiliary input is also given to both the verifier program and the simulator, however if it is sufficiently long then only the distinguisher can make any use of its suffix. It follows that the simulator guaranteed in Definition 6.18 produces output that is indistinguishable from the real interactions also by non-uniform polynomial-size machines. Namely, for every (even non-uniform) polynomial-size circuit family, $\{C_n\}_{n\in\mathbb{N}}$, every polynomial $p(\cdot)$, and all sufficiently large n's, all $x \in L \cap \{0,1\}^n$, all $y \in P_L(x)$ and $z \in \{0,1\}^*$,

$$|\mathrm{Prob}(C_n(x,z,\langle P(y),V^*(z)\rangle(x))=1) - \mathrm{Prob}(C_n(x,z,M^*(x,z))=1)| < \frac{1}{p(|x|)}$$

Following is a sketch of the proof. We assume, to the contrary, that there exists a polynomial-size circuit family, $\{C_n\}_{n\in\mathbb{N}}$, such that for infinitely many n's there exists triples (x,y,z) for which C_n has a non-negligible distinguishing gap. We derive a contradiction by incorporating the description of C_n together with the auxiliary input z into a longer auxiliary input, denoted z'. This is done in a way that both V^* and M^* have no sufficient time to reach the description of C_n. For example, let $q(\cdot)$ be a polynomial bounding the running-time of both V^* and M^*, as well as the size of C_n. Then, we let z' be the string which results by padding z with blanks to a total length of $q(n)$ and appending the description of the circuit C_n at its end (i.e., if $|z| > q(n)$ then z' is a prefix of z). Clearly, $M^*(x,z') = M^*(x,z)$ and $\langle P(y),V^*(z')\rangle(x) = \langle P(y),V^*(z)\rangle(x)$. On the other hand, by using a circuit evaluating algorithm, we get an algorithm D such that $D(x,z',\alpha) = C_n(x,z)$, and contradiction follows.

6.3.4 Sequential Composition of Zero-Knowledge Proofs

An intuitive requirement that a definition of zero-knowledge proofs must satisfy is that zero-knowledge proofs are closed under sequential composition. Namely, if one executes one zero-knowledge proof after another then the composed execution must be zero-knowledge. The same should remain valid even if one executes polynomially many proofs one after the other. Indeed, as we will shortly see, the revised definition of zero-knowledge (i.e., Definition 6.18) satisfies this requirement. Interestingly, zero-knowledge proofs as defined in Definition 6.12 are not closed under sequential composition, and this fact is indeed another indication to the necessity of augmenting this definition (as done in Definition 6.18).

In addition to its conceptual importance, the Sequential Composition Lemma is an important tool in the design of zero-knowledge proof systems. Typically, these proof system consists of many repetitions of a atomic zero-knowledge proof. Loosely speaking, the atomic proof provides some (but not much) statistical evidence to the validity of the claim. By repeating the atomic proof sufficiently many times the confidence in the validity of the claim is increased. More precisely, the atomic proof offers a gap between the accepting probability of string in the language and strings outside the language. For example, in Construction 6.16 pairs of isomorphic graphs (i.e., inputs in GI) are accepted with probability 1, whereas pairs of non-isomorphic graphs (i.e., inputs not in GI) are accepted with probability at most $\frac{1}{2}$. By repeating the atomic proof the gap between the two probabilities is further increased. For example, repeating the proof of Construction 6.16 for k times yields a new interactive proof in which inputs in GI are still accepted with probability 1 whereas inputs not in GI are accepted with probability at most $\frac{1}{2^k}$. The Sequential Composition Lemma guarantees that if the atomic proof system is zero-knowledge then so is the proof system resulting by repeating the atomic proof polynomially many times.

Before we state the Sequential Composition Lemma, we remind the reader that the zero-knowledge property of an interactive proof is actually a property of the prover. Also, the prover is required to be zero-knowledge only on inputs in the language. Finally, we stress that when talking on zero-knowledge with respect to auxiliary input we refer to all possible auxiliary inputs for the verifier.

Lemma 6.19 (Sequential Composition Lemma): *Let P be an interactive machine (i.e., a prover) which is* zero-knowledge with respect to auxiliary input *on some language L. Suppose that the last message sent by P, on input x, bears a special "end of proof" symbol. Let $Q(\cdot)$ be a polynomial, and let P_Q be an interactive machine that, on common input x, proceeds in $Q(|x|)$ phases, each of them consisting of running P on common input x. (We stress that in case P is probabilistic, the interactive machine P_Q uses independent coin tosses for each of the $Q(|x|)$ phases.) Then P_Q is zero-knowledge (with respect to auxiliary input) on L. Furthermore, if P is perfect zero-knowledge (with respect to auxiliary input) then so is P_Q.*

The convention concerning "end of proof" is introduced for technical purposes (and is redundant in all known provers for which the number of messages sent is easily computed from the length of the common input). Clearly, every machine P can be easily modified so that its last message bears an appropriate symbol (as assumed above), and doing so preserves the zero-knowledge properties of P (as well as completeness and soundness conditions).

The Lemma remain valid also if one allows auxiliary input to the prover. The extension is straightforward. The lemma ignores other aspects of repeating an interactive proof several times; specifically, the effect on the gap between the accepting probability of inputs inside and outside of the language. This aspect of repetition is discussed in the previous section (see also Exercise 1).

Proof: Let V^* be an *arbitrary* probabilistic polynomial-time interactive machine interacting with the composed prover P_Q. Our task is to construct a (polynomial-time) simulator, M^*, which simulates the real interactions of V^* with P_Q. Following is a very high level description of the simulation. The key idea is to simulate the real interaction on common input x in $Q(|x|)$ phases corresponding to the phases of the operation of P_Q. Each phase of the operation of P_Q is simulated using the simulator guaranteed for the atomic prover P. The information accumulated by the verifier in each phase is passed to the next phase using the auxiliary input.

The first step in carrying-out the above plan is to partition the execution of an arbitrary interactive machine V^* into phases. The partition may not exist in the code of the program V^*, and yet it can be imposed on the executions of this program. This is done using the phase structure of the prescribed prover P_Q, which is induced by the "end of proof" symbols. Hence, we claim that no matter how V^* operates, the interaction of V^* with P_Q on common input x, can be captured by $Q(|x|)$ successive interaction of a related machine, denoted V^{**}, with P. Namely,

Claim 6.19.1: There exists a probabilistic polynomial-time V^{**} so that for every common input x and auxiliary input z it holds that

$$\begin{aligned} \langle P_Q, V^*(z)\rangle(x) &= Z^{(Q(|x|))} \\ &\quad \text{where } Z^{(0)} \stackrel{\text{def}}{=} z \text{ and } Z^{(i+1)} \stackrel{\text{def}}{=} \langle P, V^{**}(Z^{(i)})\rangle(x) \end{aligned}$$

Namely, $Z^{(Q(|x|))}$ is a random variable describing the output of V^{**} after $Q(|x|)$ successive interactions with P, on common input x, where the auxiliary input of V^{**} in the $i+1^{\text{st}}$ interaction equals the output of V^{**} after the i^{th} interaction (i.e., $Z^{(i)}$).

proof: Consider an interaction of $V^*(z)$ with P_Q, on common input x. Machine V^* can be slightly modified so that it starts its execution by reading the common-input, the random-input and the auxiliary-input into special regions in its work-tape, and never accesses the above read-only tapes again. Likewise, V^* is modified so that it starts each active period by reading the current incoming message from the communication-tape to a special region in the work tape (and never accesses the incoming message-tape again during this period). Actually, the above description should be modified so that V^* copies only a polynomially long (in the common input) prefix of each of these tapes, the polynomial being the one bounding the running time of V^*.

Considering the contents of the work-tape of V^* at the end of each of the $Q(|x|)$ phases (of interactions with P_Q), naturally leads us to the construction of V^{**}. Namely, on common input x and auxiliary input z', machine V^{**} starts by copying z' into the work-tape of V^*. Next, machine V^{**} simulates a *single phase* of the interaction of V^* with P_Q (on input x) starting with the above contents of the work-tape of V^* (instead of starting with an empty work-tape). The invoked machine V^* regards the communication-tapes of machine V^{**} as

its own communication-tapes. Finally, V^{**} terminates by outputting the current contents of the work-tape of V^*. Actually, the above description should be slightly modified to deal differently with the first phase in the interaction with P_Q. Specifically, V^{**} copies z' into the work-tape of V^* only if z' encodes a contents of the work-tape of V^* (we assume, w.l.o.g., that the contents of the work-tape of V^* is encoded differently from the encoding of an auxiliary input for V^*). In case z' encodes an auxiliary input to V^*, machine V^{**} invokes V^* on an empty work-tape, and V^* regards the readable tapes of V^{**} (i.e., common-input-tape, the random-input-tape and the auxiliary-input-tape) as its own. Observe that $Z^{(1)} \stackrel{\text{def}}{=} \langle P, V^{**}(z)\rangle(x)$ describes the contents of the work-tape of V^* after one phase, and $Z^{(i)} \stackrel{\text{def}}{=} \langle P, V^{**}(Z^{(i-1)})\rangle(x)$ describes the contents of the work-tape of V^* after i phases. The claim follows. □

Since V^{**} is a polynomial-time interactive machine (with auxiliary input) interacting with P, it follows by the lemma's hypothesis that there exists a probabilistic machine which simulates these interactions in time polynomial in the length of the first input. Let M^{**} denote this simulator. We may assume, without loss of generality, that with overwhelmingly high probability M^{**} halts with output (as we can increase the probability of output by successive applications of M^{**}). Furthermore, for sake of simplicity, we assume in the rest of this proof that M^{**} always halts with output. Namely, for every probabilistic polynomial-time (in x) algorithm D, every polynomial $p(\cdot)$, all sufficiently long $x \in L$ and all $z \in \{0,1\}^*$, we have

$$|\mathrm{Prob}(D(x,z,\langle P, V^{**}(z)\rangle(x)) = 1) - \mathrm{Prob}(D(x,z,M^{**}(x,z)) = 1)| < \frac{1}{p(|x|)}$$

We are now ready to present the construction of a simulator, M^*, that simulates the "real" output of V^* after interaction with P_Q. Machine M^* uses the above guaranteed simulator M^{**}. On input (x,z), machine M^* sets $z^{(0)} = z$ and proceeds in $Q(|x|)$ phases. In the i^{th} phase, machine M^* computes $z^{(i)}$ by running machine M^{**} on input $(x, z^{(i-1)})$. After $Q(|x|)$ phases are completed, machine M^* stops outputting $z^{(Q(|x|))}$.

Clearly, machine M^*, constructed above, runs in time polynomial in its first input. (For non-constant $Q(\cdot)$ it is crucial here that the running-time of M^* is polynomial in the length of the first input, rather than being polynomial in the length of both inputs.) It is left to show that machine M^* indeed produces output which is polynomially indistinguishable from the output of V^* (after interacting with P_Q). Namely,

Claim 6.19.2: For every probabilistic algorithm D, with running-time polynomial in its first input, every polynomial $p(\cdot)$, all sufficiently long $x \in L$ and all $z \in \{0,1\}^*$, we have

$$|\mathrm{Prob}(D(x,z,\langle P_Q, V^*(z)\rangle(x)) = 1) - \mathrm{Prob}(D(x,z,M^*(x,z)) = 1)| < \frac{1}{p(|x|)}$$

proof sketch: We use a hybrid argument. In particular, we define the following $Q(|x|)+1$ hybrids. The i^{th} hybrid, $0 \leq i \leq Q(|x|)$, corresponds to the following random process. We first let V^{**} interact with P for i phases, starting with common input x and auxiliary input z, and denote by $Z^{(i)}$ the output of V^{**} after the i^{th} phase. We next repeatedly iterate M^{**} for the remaining $Q(m)-k$ phases. In both cases, we use the output of the previous phase as auxiliary input to the new phase. Formally, the hybrid $H^{(i)}$ is defined as follows.

$$\begin{aligned} H^{(i)}(x,z) &\stackrel{\text{def}}{=} M^{**}_{Q(m)-i}(x, Z^{(i)}) \\ &\quad \text{where } Z^{(0)} \stackrel{\text{def}}{=} z \text{ and } Z^{(j+1)} \stackrel{\text{def}}{=} \langle P, V^{**}(Z^{(j)})\rangle(x) \\ &\quad M^{**}_0(x,z') \stackrel{\text{def}}{=} (x,z') \text{ and } M^{**}_j(x,z') \stackrel{\text{def}}{=} M^{**}_{j-1}(x, M^{**}(x,z')) \end{aligned}$$

Using Claim 6.19.1, the $Q(|x|)^{\text{th}}$ hybrid (i.e., $H^{(Q(|x|))}(x,z)$) equals $\langle P_Q, V^*(z)\rangle(x))$. On the other hand, recalling the construction of M^*, we see that the zero hybrid (i.e., $H^{(0)}(x,z)$) equals $M^*(x,z)$). Hence, all that is required to complete the proof is to show that every two adjacent hybrids are polynomially indistinguishable (as this would imply that the extreme hybrids, $H^{(Q(m))}$ and $H^{(0)}$, are indistinguishable too). To this end, we rewrite the i^{th} and $i-1^{\text{st}}$ hybrids as follows.

$$\begin{aligned} H^{(i)}(x,z) &= M^{**}_{Q(|x|)-i}(x, \langle P, V^{**}(Z^{(i-1)})\rangle(x)) \\ H^{(i-1)}(x,z) &= M^{**}_{Q(|x|)-i}(x, M^{**}(x, Z^{(i-1)})) \end{aligned}$$

where $Z^{(i-1)}$ is as defined above (in the definition of the hybrids).

Using an averaging argument, it follows that if an algorithm, D, distinguishes the hybrids $H^{(i)}(x,z)$ and $H^{(i-1)}(x,z)$ then there exists a z' so that algorithm D distinguishes the random variables $M^{**}_{Q(|x|)-i}(x, \langle P, V^{**}(z')\rangle(x))$ and $M^{**}_{Q(|x|)-i}(x, M^{**}(x,z'))$ at least as well. Incorporating algorithm M^{**} into D, we get a new algorithm D', with running time polynomially related to the former algorithms, which distinguishes the random variables $(x, z', \langle P, V^{**}(z')\rangle(x))$ and $(x, z', M^{**}(x,z'))$ at least as well. (Further details are presented below.) Contradiction (to the hypothesis that M^{**} simulates (P, V^{**})) follows. □

The lemma follows. ■

Further details concerning the proof of Claim 6.19.2: The proof of Claim 6.19.2 is rather sketchy. The main thing which is missing are details concerning the way in which an algorithm contradicting the hypothesis that M^{**} is a simulator for (P, V^{**}) is derived from an algorithm contradicting the statement of Claim 6.19.2. These details are presented below, and the reader is encouraged *not* to skip them.

Let us start with the non-problematic part. We assume, to the contradiction, that there exists a probabilistic polynomial-time algorithm, D, and a polynomial $p(\cdot)$, so that

for infinitely many $x \in L$ there exists $z \in \{0,1\}^*$ such that

$$|\mathrm{Prob}(D(x,z,\langle P_Q, V^*(z)\rangle(x)) = 1) - \mathrm{Prob}(D(x,z,M^*(x,z)) = 1)| > \frac{1}{p(|x|)}$$

It follows that for every such x and z, there exists an $i \in \{1,...,Q(|x|)\}$ such that

$$|\mathrm{Prob}(D(x,z,H^{(i)}(x,z)) = 1) - \mathrm{Prob}(D(x,z,H^{(i-1)}(x,z)) = 1)| > \frac{1}{Q(|x|)\cdot p(|x|)}$$

Denote $\epsilon(n) \stackrel{\mathrm{def}}{=} 1/(Q(n)\cdot p(n))$. Combining the definition of the i^{th} and $i-1^{\mathrm{st}}$ hybrids with an averaging argument, it follows that for each such x, z and i, there exists a z', in the support of $Z^{(i-1)}$ (defined as above), such that

$$\begin{aligned}|&\mathrm{Prob}(D(x,z',M^{**}_{Q(|x|)-i}\langle P, V^{**}(z')\rangle(x)) = 1)\\ &-\mathrm{Prob}(D(x,z',M^{**}_{Q(|x|)-i}(M^{**}(x,z'))) = 1)| > \epsilon(|x|)\end{aligned}$$

This almost leads to the desired contradiction. Namely, the random variables $(x,z',\langle P, V^{**}(z')\rangle(x))$ and $(x,z',M^{**}(x,z'))$ can be distinguished using algorithms D and M^{**}, provided we "know" i. The problem is resolved using the fact, pointed out at the end of Subsection 6.3.3, that the output of M^{**} is undistinguished from the interactions of V^{**} with the prover even with respect to non-uniform polynomial-size circuits. Details follow.

We construct a polynomial-size circuit family, denoted $\{C_n\}$, which distinguishes $(x,z',\langle P, V^{**}(z'')\rangle(x))$ and $(x,z',M^{**}(x,z''))$, for the above-mentioned (x,z') pairs. On input x (supposedly in $L \cap \{0,1\}^n$) and α (supposedly in either $(x,z',\langle P, V^{**}(z'')\rangle(x))$ or $(x,z',M^{**}(x,z''))$), the circuit C_n, incorporating (the above-mentioned) i, uses algorithm M^{**} to compute $\beta = M_{Q(|x|)-i}(x,\alpha)$. Next C_n, using algorithm D, computes $\sigma = D((x,z'),\beta)$ and halts outputting σ. Contradiction (to the hypothesis that M^{**} is a simulator for (P,V^{**})) follows. □

And what about parallel composition?

Unfortunately, we cannot prove that zero-knowledge (even with respect to auxiliary input) is preserved under parallel composition. Furthermore, there exist zero-knowledge proofs that when played twice in parallel do yield knowledge (to a "cheating verifier"). For further details see Subsection 6.5.

The fact that zero-knowledge is not preserved under parallel composition of protocols is indeed bad news. One may even think that this fact is a conceptually annoying phenomenon. We disagree with this feeling. Our feeling is that the behaviour of protocols and "games" under parallel composition is, in general (i.e., not only in the context of zero-knowledge), a much more complex issue than the behaviour under sequential composition.

Furthermore, the only advantage of parallel composition over sequential composition is in efficiency. Hence, we don't consider the non-closure under parallel composition to be a conceptual weakness of the formulation of zero-knowledge. Yet, the "non-closure" of zero-knowledge motivates the search for either weaker or stronger notions which are preserved under parallel composition. For further details, the reader is referred to Sections 6.9 and 6.6.

6.4 Zero-Knowledge Proofs for NP

This section presents the main thrust of the entire chapter; namely, a method for constructing zero-knowledge proofs for *every* language in $\mathcal{NP}$. The importance of this method stems from its generality, which is the key to its many applications. Specifically, we observe that almost all statements one wish to prove in practice can be encoded as claims concerning membership in languages in $\mathcal{NP}$.

The method, for constructing zero-knowledge proofs for NP-languages, makes essential use of the concept of *bit commitment*. Hence, we start with a presentation of this concept.

6.4.1 Commitment Schemes

Commitment schemes are a basic ingredient in many cryptographic protocols. The are used to enable a party to commit itself to a value while keeping it secret. In a latter stage the commitment is "opened" and it is guaranteed that the "opening" can yield only a single value determined in the committing phase. Commitment schemes are the digital analogue of nontransparent sealed envelopes. By putting a note in such an envelope a party commits itself to the contents of the note while keeping it secret.

Definition

Loosely speaking, a commitment scheme is an efficient *two-phase* two-party protocol through which one party, called the *sender*, can commit itself to a *value* so the following two conflicting requirements are satisfied.

1. *Secrecy*: At the end of the first phase, the other party, called the *receiver*, does not gain any knowledge of the sender's value. This requirement has to be satisfied even if the receiver tries to cheat.
2. *Unambiguity*: Given the transcript of the interaction in the first phase, there exists at most one value which the receiver may later (i.e., in the second phase) accept as a legal "opening" of the commitment. This requirement has to be satisfied even if the sender tries to cheat.

In addition, one should require that the protocol is *viable* in the sense that if both parties follow it then, at the end of the second phase, the receiver gets the value committed to by the sender. The first phase is called the *commit phase*, and the second phase is called the *reveal phase*. We are requiring that the commit phase yield no knowledge (at least not of the sender's value) to the receiver, whereas the commit phase does "commit" the sender to a unique value (in the sense that in the reveal phase the receiver may accept only this value). We stress that the protocol is efficient in the sense that the predetermined programs of both parties can be implemented in probabilistic, polynomial-time. Without loss of generality, the reveal phase may consist of merely letting the sender send, to the receiver, the original value and the sequence of random coin tosses that it has used during the commit phase. The receiver will accept the value if and only if the supplied information matches its transcript of the interaction in the commit phase. The latter convention leads to the following definition (which refers explicitly only to the commit phase).

Definition 6.20 (bit commitment scheme)*: A* bit commitment scheme *is a pair of probabilistic polynomial-time interactive machines, denoted* (S, R) *(for* sender *and* receiver*), satisfying:*

- Input Specification: *The common input is an integer* n *presented in unary (serving as the security parameter). The private input to the sender is a bit* v*.*
- Secrecy: *The receiver (even when deviating arbitrarily from the protocol) cannot distinguish a commitment to* 0 *from a commitment to* 1*. Namely, for every probabilistic polynomial-time machine* R^* *interacting with* S*, the random variables describing the output of* R^* *in the two cases, namely* $\langle S(0), R^*\rangle(1^n)$ *and* $\langle S(1), R^*\rangle(1^n)$*, are polynomially-indistinguishable.*
- Unambiguity:
  Preliminaries
  - *A* receiver's view of an interaction *with the sender, denoted* $(r, \overline{m})$*, consists of the random coins used by the receiver* (r) *and the sequence of messages received from the sender* ($\overline{m}$)*.*
  - *Let* $\sigma \in \{0,1\}$*. We say that a receiver's view (of such interaction),* $(r, \overline{m})$*, is a* possible σ-commitment *if there exists a string* s *such that* $\overline{m}$ *describes the messages received by* R *when* R *uses local coins* r *and interacts with machine* S *which uses local coins* s *and has input* $(\sigma, 1^n)$*. (Using the notation of Definition 6.13, the condition may be expressed as* $\overline{m} = \mathrm{view}^{S(\sigma,1^n,s)}_{R(1^n,r)}$*.)*
  - *We say that the receiver's view* $(r, \overline{m})$ *is* ambiguous *if it is both a possible* 0*-commitment and a possible* 1*-commitment.*

> *The* unambiguity requirement *asserts that, for all but a negligible fraction of the coin tosses of the receiver, there exists no sequence of messages (from the sender) which together with these coin tosses forms an ambiguous receiver view. Namely, that for all but a negligible fraction of the* $r \in \{0,1\}^{\mathrm{poly}(n)}$ *there is no* $\overline{m}$ *such that* $(r, \overline{m})$ *is ambiguous.*

The *secrecy requirement* (above) is analogous to the definition of indistinguishability of encryptions (i.e., Definition `[missing(enc-indist.def)]`). An equivalent formulation analogous to semantic security (i.e., Definition `[missing(enc-semant.def)]`) can be presented, but is less useful in typical applications of commitment schemes. In any case, the secrecy requirement is a computational one. On the other hand, the *unambiguity requirement* has an information theoretic flavour (i.e., it does not refer to computational powers). A dual definition, requiring information theoretic secrecy and computational unfeasibility of creating ambiguities, is presented in Subsection 6.8.2.

The secrecy requirement refers explicitly to the situation at the end of the commit phase. On the other hand, we stress that the unambiguity requirement implicitly assumes that the reveal phase takes the following form:

1. the sender sends to the receiver its initial private input, v, and the random coins, s, it has used in the commit phase;
2. the receiver verifies that v and s (together with the coins (r) used by R in the commit phase) indeed yield the messages that R has received in the commit phase. Verification is done in polynomial-time (by running the programs S and R).

Note that the viability requirement (i.e., asserting that if both parties follow the protocol then, at the end of the reveal phase, the receiver gets v) is implicitly satisfied by the above convention.

Construction based on any one-way permutation

Some public-key encryption scheme can be used as a commitment scheme. This can be done by having the sender generate a pair of keys and use the public-key together with the encryption of a value as its commitment to the value. In order to satisfy the unambiguity requirement, the underlying public-key scheme needs to satisfy additional requirements (e.g., the set of legitimate public-keys should be efficiently recognizable). In any case, public-key encryption schemes have additional properties not required of commitment schemes and their existence seems to require stronger intractability assumptions. An alternative construction, presented below, uses any one-way permutation. Specifically, we use a one-way permutation, denoted f, and a hard-core predicate for it, denoted b (see Section 2.5).

Construction 6.21 (simple bit commitment): *Let $f : \{0,1\}^* \mapsto \{0,1\}^*$ be a function, and $b : \{0,1\}^* \mapsto \{0,1\}$ be a predicate.*

1. commit phase: *To commit to value $v \in \{0,1\}$ (using security parameter n), the sender uniformly selects $s \in \{0,1\}^n$ and sends the pair $(f(s), b(s) \oplus v)$ to the receiver.*
2. reveal phase: *In the reveal phase, the sender reveals the string s used in the commit phase. The receiver accepts the value v if $f(s) = \alpha$ and $b(s) \oplus v = \sigma$, where (α, σ) is the receiver's view of the commit phase.*

Proposition 6.22 *Let $f : \{0,1\}^* \mapsto \{0,1\}^*$ be a length preserving 1-1 one-way function, and $b : \{0,1\}^* \mapsto \{0,1\}$ be a hard-core predicate of f. Then, the protocol presented in Construction 6.21 constitutes a bit commitment scheme.*

Proof: The secrecy requirement follows directly from the fact that b is a hard-core of f. The unambiguity requirement follows from the 1-1 property of f. In fact, there exists *no* ambiguous receiver view. Namely, for each receiver view (α, σ), there is a unique $s \in \{0,1\}^{|\alpha|}$ so that $f(s) = \alpha$ and hence a unique $v \in \{0,1\}$ so that $b(s) \oplus v = \sigma$. ■

Construction based on any one-way function

We now present a construction of a bit commitment scheme which is based on the weakest assumption possible: the existence of one-way function. Proving the that the assumption is indeed minimal is left as an exercise (i.e., Exercise 12). On the other hand, by the results in Chapter 3 (specifically, Theorems 3.11 and 3.29), the existence of one-way functions imply the existence of pseudorandom generators expanding n-bit strings into $3n$-bit strings. We will use such a pseudorandom generator in the construction presented below.

We start by motivating the construction. Let G be a pseudorandom generator satisfying $|G(s)| = 3 \cdot |s|$. Assume that G has the property that the sets $\{G(s) : s \in \{0,1\}^n\}$ and $\{G(s) \oplus 1^{3n} : s \in \{0,1\}^n\}$ are disjoint, were $\alpha \oplus \beta$ denote the bit-by-bit exclusive-or of the strings α and β. Then, the sender may commit itself to the bit v by uniformly selecting $s \in \{0,1\}^n$ and sending the message $G(s) \oplus v^{3n}$ (v^k denotes the all-v's k-bit long string). Unfortunately, the above assumption cannot be justified, in general, and a slightly more complex variant is required. The key observation is that for most strings $\beta \in \{0,1\}^{3n}$ the sets $\{G(s) : s \in \{0,1\}^n\}$ and $\{G(s) \oplus \beta : s \in \{0,1\}^n\}$ are disjoint. Such a string β is called *good*. This observation suggests the following protocol. The receiver uniformly selects $\beta \in \{0,1\}^{3n}$, hoping that it is good, and the sender commits to the bit v by uniformly selecting $s \in \{0,1\}^n$ and sending the message $G(s)$ if $v = 0$ and $G(s) \oplus \beta$ otherwise.

Construction 6.23 (bit commitment under general assumptions): *Let $G : \{0,1\}^* \mapsto \{0,1\}^*$ be a function so that $|G(s)| = 3 \cdot |s|$ for all $s \in \{0,1\}^*$.*

1. commit phase: *To receive a commitment to a bit (using security parameter n), the receiver uniformly selects $r \in \{0,1\}^{3n}$ and sends it to the sender. Upon receiving the message r (from the receiver), the sender commits to value $v \in \{0,1\}$ by uniformly selecting $s \in \{0,1\}^n$ and sending $G(s)$ if $v = 0$ and $G(s) \oplus r$ otherwise.*

2. reveal phase: *In the reveal phase, the sender reveals the string s used in the commit phase. The receiver accepts the value 0 if $G(s) = \alpha$ and the value 1 if $G(s) \oplus r = \alpha$, where (r, α) is the receiver's view of the commit phase.*

Proposition 6.24 *If G is a pseudorandom generator, then the protocol presented in Construction 6.23 constitutes a bit commitment scheme.*

Proof: The secrecy requirement follows the fact that G is a pseudorandom generator. Specifically, let U_k denote the random variable uniformly distributed on strings of length k. Then for every $r \in \{0,1\}^{3n}$, the random variables U_{3n} and $U_{3n} \oplus r$ are identically distributed. Hence, if it is feasible to find an $r \in \{0,1\}^{3n}$ such that $G(U_n)$ and $G(U_n) \oplus r$ are computationally distinguishable then either U_{3n} and $G(U_n)$ are computationally distinguishable or $U_{3n} \oplus r$ and $G(U_n) \oplus r$ are computationally distinguishable. In either case contradiction to the pseudorandomness of G follows.

We now turn to the unambiguity requirement. Following the motivating discussion, we call $\beta \in \{0,1\}^{3n}$ *good* if the sets $\{G(s) : s \in \{0,1\}^n\}$ and $\{G(s) \oplus \beta : s \in \{0,1\}^n\}$ are disjoint. We say that $\beta \in \{0,1\}^{3n}$ *yields a collision between the seeds s_1 and s_2* if $G(s_1) = G(s_2) \oplus \beta$. Clearly, β is good if it does not yield a collision between any pair of seeds. On the other hand, there is a unique string β which yields a collision between a given pair of seeds (i.e., $\beta = G(s_1) \oplus G(s_2)$). Since there are 2^{2n} possible pairs of seeds, at most 2^{2n} strings yield collisions between seeds and all the other $3n$-bit long strings are good. It follows that with probability at least $1 - 2^{2n-3n}$ the receiver selects a good string. The unambiguity requirement follows. ■

Extensions

The definition and the constructions of bit commitment schemes are easily extended to general commitment schemes enabling the sender to commit to a string rather than to a single bit. When defining the secrecy of such schemes the reader is advised to consult Definition `[missing(enc-indist.def)]`). For the purposes of the rest of this section we need a commitment scheme by which one can commit to a ternary value. Extending the definition and the constructions to deal with this case is even more straightforward.

In the rest of this section we will need commitment schemes with a seemingly stronger secrecy requirement than defined above. Specifically, instead of requiring secrecy with

respect to all polynomial-time machines, we will require secrecy with respect to all (not necessarily uniform) families of polynomial-size circuits. Assuming the existence of non-uniformly one-way functions (see Definition 2.6 in Section 2.2) commitment schemes with nonuniform secrecy can be constructed, following the same constructions used in the uniform case.

6.4.2 Zero-Knowledge proof of Graph Coloring

Presenting a zero-knowledge proof system for one $\mathcal{NP}$-complete language implies the existence of a zero-knowledge proof system for every language in $\mathcal{NP}$. This intuitively appealing statement does require a proof which we postpone to a later stage. In the current subsection we present a zero-knowledge proof system for one $\mathcal{NP}$-complete language, specifically Graph 3-Colorability. This choice is indeed arbitrary.

The language *Graph 3-Coloring*, denoted $G3C$, consists of all simple graphs (i.e., no parallel edges or self-loops) that can be *vertex-colored* using 3 colors so that no two adjacent vertices are given the same color. Formally, a graph $G=(V,E)$, is *3-colorable*, if there exists a mapping $\phi : V \mapsto \{1,2,3\}$ so that $\phi(u) \neq \phi(v)$ for every $(u,v) \in E$.

Motivating discussion

The idea underlying the zero-knowledge proof system for $G3C$ is to break the proof of the claim that a graph is 3-colorable into polynomially many *pieces* arranged in *templates* so that each template by itself yields no knowledge and yet all the templates put together guarantee the validity of the main claim. Suppose that the prover generates such pieces of information, places each of them in a separate sealed and nontransparent envelope, and allows the verifier to open and inspect the pieces participating in one of the templates. Then certainly the verifier gains no knowledge in the process, yet his confidence in the validity of the claim (that the graph is 3-colorable) increases. A concrete implementation of this abstract scheme follows.

To prove that the graph $G = (V,E)$ is 3-colorable, the prover generates a random 3-coloring of the graph, denoted ϕ (actually a random relabelling of a fixed coloring will do). The color of each single vertex constitutes a piece of information concerning the 3-coloring. The set of templates corresponds to the set of edges (i.e., each pair $(\phi(u),\phi(v)), (u,v) \in E$, constitutes a template to the claim that G is 3-colorable). Each single template (being merely a random pair of distinct elements in $\{1,2,3\}$) yield no knowledge. However, if all the templates are OK then the graph must be 3-colorable. Consequently, graphs which are not 3-colorable must contain at least one bad template and hence are rejected with non-negligible probability. Following is an abstract description of the resulting zero-knowledge interactive proof system for $G3C$.

- *Common Input:* A simple graph $G=(V,E)$.
- *Prover's first step:* Let ψ be a 3-coloring of G. The prover selects a random permutation, π, over $\{1,2,3\}$, and sets $\phi(v) \stackrel{\text{def}}{=} \pi(\psi(v))$, for each $v \in V$. Hence, the prover forms a random relabelling of the 3-coloring ψ. The prover sends the verifier a sequence of $|V|$ locked and nontransparent boxes so that the v^{th} box contains the value $\phi(v)$;
- *Verifier's first step:* The verifier uniformly selects an edge $(u,v) \in E$, and sends it to the prover;
- *Motivating Remark:* The verifier asks to inspect the colors of vertices u and v;
- *Prover's second step:* The prover sends to the verifier the keys to boxes u and v;
- *Verifier's second step:* The verifier opens boxes u and v, and accepts if and only if they contain two different elements in $\{1,2,3\}$;

Clearly, if the input graph is 3-colorable then the prover can cause the verifier to accept always. On the other hand, if the input graph is not 3-colorable then any contents placed in the boxes must be invalid on at least one edge, and consequently the verifier will reject with probability at least $1/|E|$. Hence, the above protocol exhibits a non-negligible gap in the accepting probabilities between the case of inputs in $G3C$ and inputs not in $G3C$. The zero-knowledge property follows easily, in this abstract setting, since one can simulate the real interaction by placing a random pair of different colors in the boxes indicated by the verifier. We stress that this simple argument will not be possible in the digital implementation since the boxes are not totally ineffected by their contents (but are rather effected, yet in an indistinguishable manner). Finally, we remark that the confidence in the validity of the claim (that the input graph is 3-colorable) may be increased by sequentially applying the above proof sufficient many times. (In fact if the boxes are perfect as assumed above then one can also use parallel repetitions.)

The interactive proof

We now turn to the digital implementation of the above abstract protocol. In this implementation the boxes are implemented by a commitment scheme. Namely, for each box we invoke an independent execution of the commitment scheme. This will enable us to execute the reveal phase in only some of the commitments, a property that is crucial to our scheme. For simplicity of exposition, we use the simple commitment scheme presented in Construction 6.21 (or, more generally, any *one-way interaction* commitment scheme). We denote by $C_s(\sigma)$ the commitment of the sender, using coins s, to the (ternary) value σ.

Construction 6.25 (A zero-knowledge proof for Graph 3-Coloring):

- Common Input: *A simple (3-colorable) graph $G = (V, E)$. Let $n \stackrel{\text{def}}{=} |V|$ and $V = \{1, ..., n\}$.*
- Auxiliary Input to the Prover: *A 3-coloring of G, denoted ψ.*
- Prover's first step (P1): *The prover selects a random permutation, π, over $\{1, 2, 3\}$, and sets $\phi(v) \stackrel{\text{def}}{=} \pi(\psi(v))$, for each $v \in V$. The prover uses the commitment scheme to commit itself to the color of each of the vertices. Namely, the prover uniformly and independently selects $s_1, ..., s_n \in \{0, 1\}^n$, computes $c_i = C_{s_i}(\phi(i))$, for each $i \in V$, and sends $c_1, ..., c_n$ to the verifier;*
- Verifier's first step (V1): *The verifier uniformly selects an edge $(u, v) \in E$, and sends it to the prover;*
- Motivating Remark: *The verifier asks to inspect the colors of vertices u and v;*
- Prover's second step (P2): *Without loss of generality, we may assume that the message received for the verifier is an edge, denoted (u, v). (Otherwise, the prover sets (u, v) to be some predetermined edge of G.) The prover uses the reveal phase of the commitment scheme in order to reveal the colors of vertices u and v to the verifier. Namely, the prover sends $(s_u, \phi(u))$ and $(s_v, \phi(v))$ to the verifier;*
- Verifier's second step (V2): *The verifier checks whether the values corresponding to commitments u and v were revealed correctly and whether these values are different. Namely, upon receiving (s, σ) and (s', τ), the verifier checks whether $c_u = C_s(\sigma)$, $c_v = C_{s'}(\tau)$, and $\sigma \neq \tau$ (and both in $\{1, 2, 3\}$). If all conditions hold then the verifier accepts. Otherwise it rejects.*

Let us denote the above prover's program by P_{G3C}.

We stress that both the programs of the verifier and of the prover can be implemented in probabilistic polynomial-time. In case of the prover's program this property is made possible by the use of the auxiliary input to the prover. As we will shortly see, the above protocol constitutes a weak interactive proof for $G3C$. As usual, the confidence can be increased (i.e., the error probability can be decreased) by sufficiently many successive applications. However, the mere existence of an interactive proof for $G3C$ is obvious (since $G3C \in \mathcal{NP}$). The punch-line is that the above protocol is zero-knowledge (also with respect to auxiliary input). Using the Sequential Composition Lemma (Lemma 6.19), it follows that also polynomially many sequential applications of this protocol preserve the zero-knowledge property.

Proposition 6.26 *Suppose that the commitment scheme used in Construction 6.25 satisfies the (nonuniform) secrecy and the unambiguity requirements. Then Construction 6.25 constitutes an auxiliary input zero-knowledge (generalized) interactive proof for $G3C$.*

For further discussion of Construction 6.25 see remarks at the end of the current subsection.

Proof of Proposition 6.26

We first prove that Construction 6.25 constitutes a weak interactive proof for $G3C$. Assume first that the input graph is indeed 3-colorable. Then if the prover follows the program in the construction then the verifier will always accept (i.e., accept with probability 1). On the other hand, if the input graph is not 3-colorable then, no matter what the prover does, the n commitments sent in Step (P1) cannot "correspond" to a 3-coloring of the graph (since such coloring does not exists). We stress that the unique correspondence of commitments to values is guaranteed by the unambiguity property of the commitment scheme. It follows that there must exists an edge $(u,v) \in E$ so that c_u and c_v, sent in step (P1), are not commitments to two different elements of $\{1,2,3\}$. Hence, no matter how the prover behaves, the verifier will reject with probability at least $1/|E|$. Hence there is a non-negligible (in the input length) gap between the accepting probabilities in case the input is in $G3C$ and in case it is not.

We now turn to show that P_{G3C}, the prover in Construction 6.25, is indeed zero-knowledge for $G3C$. The claim is proven without reference to auxiliary input (to the verifier), yet extending the argument to auxiliary input zero-knowledge is straightforward. Again, we will use the alternative formulation of zero-knowledge (i.e., Definition 6.13), and show how to simulate V^*'s view of the interaction with P_{G3C}, for every probabilistic polynomial-time interactive machine V^*. As in the case of the Graph Isomorphism proof system (i.e., Construction 6.16) it is quite easy to simulate the verifier's view of the interaction with P_{G3C}, *provided that* the verifier follows the specified program. However, we need to simulate the view of the verifier in the general case (in which it uses an arbitrary polynomial-time interactive program). Following is an overview of our simulation (i.e., of our construction of a simulator, M^*, for an arbitrary V^*).

The simulator M^* incorporates the code of the interactive program V^*. On input a graph $G=(V,E)$, the simulator M^* (not having access to a 3-coloring of G) first uniformly and independently selects n values $e_1,...,e_n \in \{1,2,3\}$, and constructs a commitment to each of them. These e_i's constitute a "pseudo-coloring" of the graph, in which the end-points of each edge are colored differently with probability $\frac{2}{3}$. In doing so, the simulator behaves very differently from P_{G3C}, but nevertheless the sequence of commitments so generated is computationally indistinguishable from the sequence of commitments to a valid 3-coloring sent by P_{G3C} in step (P1). If V^*, when given the commitments generated by the simulator, asks to inspect an edge (u,v) so that $e_u \neq e_v$ then the simulator can indeed answer correctly, and doing so it completes a simulation of the verifier's view of the interaction with P_{G3C}. However, if V^* asks to inspect an edge (u,v) so that $e_u = e_v$ then the simulator has no way to answer correctly, and we let it halt with output $\perp$. We stress that we don't assume that the simulator a-priori "knows" which edge the verifier V^* will ask to inspect. The validity

of the simulator stems from a different source. If the verifier's request were oblivious of the prover's commitment then with probability $\frac{2}{3}$ the verifier would have asked to inspect an edge which is properly colored. Using the secrecy property of the commitment scheme it follows that the verifier's request is "almost oblivious" of the values in the commitments. The zero-knowledge claim follows (yet, with some effort). Further detail follow. We start with a detailed description of the simulator.

Simulator M^*. On input a graph $G=(V,E)$, the simulator M^* proceeds as follows:

1. *Setting the random tape of* V^*: Let $q(\cdot)$ denote a polynomial bounding the running-time of V^*. The simulator M^* starts by uniformly selecting a string $r \in \{0,1\}^{q(|x|)}$, to be used as the contents of the local random tape of V^*.

2. *Simulating the prover's first step (P1):* The simulator M^* uniformly and independently selects n values $e_1, ..., e_n \in \{1,2,3\}$ and n random strings $s_1, ..., s_n \in \{0,1\}^n$ to be used for committing to these values. The simulator computes, for each $i \in V$, a commitment $d_i = C_{s_i}(e_i)$.

3. *Simulating the verifier's first step (V1):* The simulator M^* initiates an execution of V^* by placing G on V^*'s "common input tape", placing r (selected in step (1) above) on V^*'s "local random tape", and placing the sequence $(d_1, ..., d_n)$ (constructed in step (2) above) on V^*'s "incoming message tape". After executing a polynomial number of steps of V^*, the simulator can read the outgoing message of V^*, denoted m. Again, we assume without loss of generality that $m \in E$ and let $(u,v) = m$. (Actually $m \notin E$ is treated as in step (P2) in P_{G3C}; namely, (u,v) is set to be some predetermined edge of G.)

4. *Simulating the prover's second step (P2):* If $e_u \neq e_v$ then the simulator halts with output $(G, r, (d_1, ..., d_n), (s_u, e_u, s_v, e_v))$.

5. *Failure of the simulation:* Otherwise (i.e., $e_u = e_v$), the simulator halts with output $\perp$.

Using the hypothesis that V^* is polynomial-time, it follows that so is the simulator M^*. It is left to show that M^* outputs $\perp$ with probability at most $\frac{1}{2}$, and that, conditioned on not outputting $\perp$, the simulator's output is computationally indistinguishable from the verifier's view in a "real interaction with P_{G3C}". The proposition will follow by running the above simulator n times and outputting the first output different from $\perp$. We now turn to prove the above two claims.

Claim 6.26.1: For every sufficiently large graph, $G=(V,E)$, the probability that $M^*(G) = \perp$ is bounded above by $\frac{1}{2}$.

proof: As above, n will denote the cardinality of the vertex set of G. Let us denote by $p_{u,v}(G, r, (e_1, ..., e_n))$ the probability, taken over all the choices of the $s_1, ..., s_n \in \{0,1\}^n$, that V^*, on input G, random coins r, and prover message $(C_{s_1}(e_1), ..., C_{s_n}(e_n))$, replies with the message (u, v). We assume, for simplicity, that V^* always answers with an edge of G (since otherwise its message is anyhow treated as if it were an edge of G). We first claim that for every sufficiently large graph, $G=(V,E)$, every $r \in \{0,1\}^{q(n)}$, every edge $(u,v) \in E$, and every two sequences $\alpha, \beta \in \{1,2,3\}^n$, it holds that

$$|p_{u,v}(G, r, \alpha) - p_{u,v}(G, r, \beta)| \leq \frac{1}{2|E|}$$

Actually, we can prove the following.

Request Obliviousness Subclaim: For every polynomial $p(\cdot)$, every sufficiently large graph, $G = (V,E)$, every $r \in \{0,1\}^{q(n)}$, every edge $(u,v) \in E$, and every two sequences $\alpha, \beta \in \{1,2,3\}^n$, it holds that

$$|p_{u,v}(G, r, \alpha) - p_{u,v}(G, r, \beta)| \leq \frac{1}{p(n)}$$

The Request Obliviousness Subclaim is proven using the non-uniform secrecy of the commitment scheme. The reader should be able to fill-up the details of such a proof at this stage. Nevertheless, a proof of the subclaim follows.

> *Proof of the Request Obliviousness Subclaim*: Assume on the contrary that there exists a polynomial $p(\cdot)$, and an infinite sequence of integers such that for each integer n (in the sequence) there exists an n-vertices graph, $G_n = (V_n, E_n)$, a string $r_n \in \{0,1\}^{q(n)}$, an edge $(u_n, v_n) \in E_n$, and two sequences $\alpha_n, \beta_n \in \{1,2,3\}^n$ so that
>
> $$|p_{u_n,v_n}(G_n, r_n, \alpha_n) - p_{u_n,v_n}(G_n, r_n, \beta_n)| > \frac{1}{p(n)}$$
>
> We construct a circuit family, $\{A_n\}$, by letting A_n incorporate the interactive machine V^*, the graph G_n, and $r_n, u_n, v_n, \alpha_n, \beta_n$, all being as in the contradiction hypothesis. On input, y (supposedly a commitment to either α_n or β_n), circuit A_n runs V^* (on input G_n coins r_n and prover's message y), and outputs 1 if and only if V^* replies with (u_n, v_n). Clearly, $\{A_n\}$ is a (non-uniform) family of polynomial-size circuits. The key observation is that A_n distinguishes commitments to α_n from commitments to β_n, since
>
> $$\text{Prob}(A_n(C_{U_{n^2}}(\gamma)) = 1) = p_{u_n,v_n}(G_n, r_n, \gamma)$$
>
> where U_k denotes, as usual, a random variable uniformly distributed over $\{0,1\}^k$. Contradiction to the (non-uniform) secrecy of the commitment scheme follows by a standard hybrid argument (which relates the indistinguishability of sequences to the indistinguishability of single commitments).

Returning to the proof of Claim 6.26.1, we now use the above subclaim to upper bound the probability that the simulator outputs $\perp$. The intuition is simple. Since the requests of V^* are almost oblivious of the values to which the simulator has committed itself, it is unlikely that V^* will request to inspect an illegally colored edge more often than if he would have made the request without looking at the commitment. A formal (but straightforward) analysis follows.

Let $M^*_r(G)$ denote the output of machine M^* on input G, conditioned on the event that it chooses the string r in step (1). We remind the reader that $M^*_r(G) = \perp$ only in case the verifier on input G, random tape r, and a commitment to some pseudo-coloring $(e_1, ..., e_n)$, asks to inspect an edge (u, v) which is illegally colored (i.e., $e_u = e_v$). Let $E_{(e_1,...,e_n)}$ denote the set of edges $(u, v) \in E$ that are illegally colored (i.e., satisfy $e_u = e_v$) with respect to $(e_1, ..., e_n)$. Then, fixing an arbitrary r and considering all possible choices of $(e_1, ..., e_n) \in \{1, 2, 3\}^n$,

$$\text{Prob}(M^*_r(G) = \perp) = \sum_{\overline{e} \in \{1,2,3\}^n} \frac{1}{3^n} \cdot \sum_{(u,v) \in E_{\overline{e}}} p_{u,v}(G, r, \overline{e})$$

(Recall that $p_{u,v}(G, r, \overline{e})$ denotes the probability that the verifier asks to inspect (u, v) when given a sequence of random commitments to the values $\overline{e}$.) Define $B_{u,v}$ to be the set of n-tuples $(e_1, ..., e_n) \in \{1, 2, 3\}^n$ satisfying $e_u = e_v$. Clearly, $|B_{u,v}| = 3^{n-1}$. By straightforward calculation we get

$$\begin{aligned}
\text{Prob}(M^*_r(G) = \perp) &= \frac{1}{3^n} \cdot \sum_{(u,v) \in E} \sum_{\overline{e} \in B_{u,v}} p_{u,v}(G, r, \overline{e}) \\
&\leq \frac{1}{3^n} \cdot \sum_{(u,v) \in E} |B_{u,v}| \cdot \left(p_{u,v}(G, r, (1, ..., 1)) + \frac{1}{2|E|} \right) \\
&= \frac{1}{6} + \frac{1}{3} \cdot \sum_{(u,v) \in E} p_{u,v}(G, r, (1, ..., 1)) \\
&= \frac{1}{6} + \frac{1}{3}
\end{aligned}$$

The claim follows. □

For simplicity, we assume in the sequel that on common input $G \in G3C$, the prover gets the *lexicographically first* 3-coloring of G as auxiliary input. This enables us to omit the auxiliary input to P_{G3C} (which is now implicit in the common input) from the notation. The argument is easily extended to the general case where P_{G3C} gets an arbitrary 3-coloring of G as auxiliary input.

Claim 6.26.2: The ensemble consisting of the output of M^* on input $G = (V, E) \in G3C$, conditioned on it not being $\perp$, is computationally indistinguishable from the ensemble

$\{\text{view}_{V^*}^{P_{G3C}}(G)\}_{G\in G3C}$. Namely, for every probabilistic polynomial-time algorithm, A, every polynomial $p(\cdot)$, and all sufficiently large graph $G=(V,E)$,

$$|\text{Prob}(A(M^*(G))=1|M^*(G)\neq\perp)-\text{Prob}(A(\text{view}_{V^*}^{P_{G3C}}(G))=1)|<\frac{1}{p(|V|)}$$

We stress that these ensembles are very different (i.e., the statistical distance between them is very close to the maximum possible), and yet they are computationally indistinguishable. Actually, we can prove that these ensembles are indistinguishable also by (non-uniform) families of polynomial-size circuits. In first glance it seems that Claim 6.26.2 follows easily from the secrecy property of the commitment scheme. Indeed, Claim 6.26.2 is proven using the secrecy property of the commitment scheme, yet the proof is more complex than one anticipates (at first glance). The difficulty lies in the fact that the above ensembles consist not only of commitments to values, but also of an opening of some of the values. Furthermore, the choice of which commitments are to be opened depends on the entire sequence of commitments.

proof: Given a graph $G=(V,E)$, we define for each edge $(u,v)\in E$ two random variables describing, respectively, the output of M^* and the view of V^* in a real interaction, in case the verifier asked to inspect the edge (u,v). Specifically

- $\mu_{u,v}(G)$ describes $M^*(G)$ conditioned on $M^*(G)$ containing the "reveal information" for vertices u and v.
- $\nu_{u,v}(G)$ describes $\text{view}_{V^*}^{P_{G3C}}(G)$ conditioned on $\text{view}_{V^*}^{P_{G3C}}(G)$ containing the "reveal information" for vertices u and v.

Let $p_{u,v}(G)$ denote the probability that $M^*(G)$ contains "reveal information" for vertices u and v, conditioned on $M^*(G)\neq\perp$. Similarly, let $q_{u,v}(G)$ denote the probability that $\text{view}_{V^*}^{P_{G3C}}(G)$ contains "reveal information" for vertices u and v.

Assume, in the contrary to the claim, that the ensembles mentioned in the claim are computationally distinguishable. Then one of the following cases must occur.

Case 1: There is a noticeable difference between the probabilistic profile of the requests of V^* when interacting with $PG3C$ and the requests of V^* when invoked by M^*. Formally, there exists a polynomial $p(\cdot)$ and an infinite sequence of integers such that for each integer n (in the sequence) there exists an n-vertices graph $G_n=(V_n,E_n)$, and an edge $(u_n,v_n)\in E_n$, so that

$$|p_{u_n,v_n}(G_n)-q_{u_n,v_n}(G_n)|>\frac{1}{p(n)}$$

Case 2: An algorithm distinguishing the above ensembles does so also conditioned on V^* asking for a particular edge. Furthermore, this request occurs with noticeable probability which is about the same in both ensembles. Formally, there exists a probabilistic polynomial-time algorithm A, a polynomial $p(\cdot)$ and an infinite sequence of integers such that for each integer n (in the sequence) there exists an n-vertices graph $G_n = (V_n, E_n)$, and an edge $(u_n, v_n) \in E_n$, so that the following conditions hold

- $q_{u_n,v_n}(G_n) > \frac{1}{p(n)}$
- $|p_{u_n,v_n}(G_n) - q_{u_n,v_n}(G_n)| < \frac{1}{3 \cdot p(n)^2}$
- $|\mathrm{Prob}(A(\mu_{u_n,v_n}(G_n)) = 1) - \mathrm{Prob}(A(\nu_{u_n,v_n}(G_n)) = 1)| > \frac{1}{p(|V|)}$.

Case 1 can be immediately discarded since it leads easily to contradiction (to the non-uniform secrecy of the commitment scheme). The idea is to use the Request Obliviousness Subclaim appearing in the proof of Claim 6.26.1. Details are omitted. We are thus left with Case 2.

We are now going to show that also Case 2 leads to contradiction. To this end we will construct a circuit family that will distinguish commitments to different sequences of values. Interestingly, neither of these sequences will equal the sequence of commitments generated by either the prover or by the simulator. Following is an overview of the construction. The n^{th} circuit gets a sequence of $3n$ commitments and produces from it a sequence of n commitments (part of which is a subsequence of the input). When the input sequence to the circuit is taken from one distribution the circuit generates a subsequence corresponding to the sequence of commitments generated by the prover. Likewise, when the input sequence (to the circuit) is taken from the other distribution the circuit will generate a subsequence corresponding to the sequence of commitments generated by the simulator. We stress that the circuit does so without knowing from which distribution the input is taken. After generated an n-long sequence, the circuit feeds it to V^*, and depending on V^*'s behaviour the circuit may feed part of the sequence to algorithm A (mentioned in Case 2). Following is a detailed description of the circuit family.

Let us denote by ψ_n the (lexicographically first) 3-coloring of G_n used by the prover. We construct a circuit family, denoted $\{A_n\}$, by letting A_n incorporate the interactive machine V^*, the "distinguishing" algorithm A, the graph G_n, the 3-coloring ψ_n, and the edge (u_n, v_n), all being those guaranteed in Case 2. The input to circuit A_n will be a sequence of commitments to $3n$ values, each in $\{1, 2, 3\}$. The circuit will distinguish commitments to a uniformly chosen $3n$-long sequence from commitments to the fixed sequence $1^n 2^n 3^n$ (i.e., the sequence consisting of n 1-values, followed by n 2-values, followed by n 3-values). Following is a description of the operation of A_n.

On input, $y = (y_1, ..., y_{3n})$ (where each y_i is supposedly a commitment to an element of $\{1, 2, 3\}$), the circuit A_n proceeds as follows.

- A_n first selects uniformly a permutation π over $\{1,2,3\}$, and computes $\phi(i) = \pi(\psi_n(i))$, for each $i \in V_n$.
- For each $i \in V_n - \{u_n, v_n\}$, the circuit sets $c_i = y_{\phi(i)\cdot n - n + i}$ (i.e., $c_i = y_i$ if $\phi(i) = 1$, $c_i = y_{n+i}$ if $\phi(i) = 2$, and $c_i = y_{2n+i}$ if $\phi(i) = 3$). Note that each y_j is used at most once, and $2n+2$ of the y_j's are not used at all.
- The circuit uniformly selects $s_u, s_v \in \{0,1\}^n$, and sets $c_{u_n} = C_{s_{u_n}}(\phi(u_n))$ and $c_{v_n} = C_{s_{v_n}}(\phi(v_n))$.
- The circuit initiates an execution of V^* by placing G_n on V^*'s "common input tape", placing a uniformly selected $r \in \{0,1\}^{q(n)}$ on V^*'s "local random tape", and placing the sequence $(c_1, ..., c_n)$ (constructed above) on V^*'s "incoming message tape". The circuit reads the outgoing message of V^*, denoted m.
- If $m \neq (u_n, v_n)$ then the circuit outputs 1.
- Otherwise (i.e., $m = (u_n, v_n)$), the circuit invokes algorithm A and outputs

$$A(G_n, r, (c_1, ..., c_n), (s_{u_n}, \phi(u_n), s_{v_n}, \phi(v_n)))$$

Clearly the size of A_n is polynomial in n. We now evaluate the distinguishing ability of A_n. Let us first consider the probability that circuit A_n outputs 1 on input a random commitment to the sequence $1^n 2^n 3^n$. The reader can easily verify that the sequence $(c_1, ..., c_n)$ constructed by circuit A_n is distributed identically to the sequence sent by the prover in step (P1). Hence, letting $C(\gamma)$ denote a random commitment to a sequence $\gamma \in \{1,2,3\}^*$, we get

$$\begin{aligned} \mathrm{Prob}(A_n(C(1^n 2^n 3^n)) = 1) &= (1 - q_{u_n,v_n}(G_n)) \\ &\quad + q_{u_n,v_n}(G_n) \cdot \mathrm{Prob}(A(\nu_{u_n,v_n}(G_n)) = 1) \end{aligned}$$

On the other hand, we consider the probability that circuit A_n outputs 1 on input a random commitment to a uniformly chosen $3n$-long sequence over $\{1,2,3\}$. The reader can easily verify that the sequence $(c_1, ..., c_n)$ constructed by circuit A_n is distributed identically to the sequence $(d_1, ..., d_n)$ generated by the simulator in step (2), conditioned on $d_{u_n} \neq d_{v_n}$. Letting T_{3n} denote a random variable uniformly distributed over $\{1,2,3\}^{3n}$, we get

$$\begin{aligned} \mathrm{Prob}(A_n(C(T_{3n}) = 1) &= (1 - p_{u_n,v_n}(G_n)) \\ &\quad + p_{u_n,v_n}(G_n) \cdot \mathrm{Prob}(A(\mu_{u_n,v_n}(G_n)) = 1) \end{aligned}$$

Using the conditions of Case 2, and omitting G_n from the notation, we get

$$|\mathrm{Prob}(A_n(C(1^n 2^n 3^n)) = 1) - \mathrm{Prob}(A_n(C(T_{3n}) = 1)|$$

$$\begin{aligned}
&\geq q_{u_n,v_n} \cdot |\mathrm{Prob}(A(\nu_{u_n,v_n}) = 1) - \mathrm{Prob}(A(\mu_{u_n,v_n}) = 1)| - 2 \cdot |p_{u_n,v_n} - q_{u_n,v_n}| \\
&> \frac{1}{p(n)} \cdot \frac{1}{p(n)} - 2 \cdot \frac{1}{3 \cdot p(n)^2} \\
&= \frac{1}{3 \cdot p(n)^2}
\end{aligned}$$

Hence, the circuit family $\{A_n\}$ distinguishes commitments to $\{1^n 2^n 3^n\}$ from commitments to $\{T_{3n}\}$. Combining an averaging argument with a hybrid argument, we conclude that there exists a polynomial-size circuit family which distinguishes commitments. This contradicts the non-uniform secrecy of the commitment scheme.

Having reached contradiction in both cases, Claim 6.26.2. □

Combining Claims 6.26.1 and 6.26.2, the zero-knowledge property of P_{G3C} follows. This completes the proof of the proposition. ■

Concluding remarks

Construction 6.25 has been presented using a unidirectional commitment scheme. A fundamental property of such schemes is that their secrecy is preserved also in case (polynomially) many instances are invoked simultaneously. The proof of Proposition 6.26 indeed took advantage on this property. We remark that Construction 6.23 also possesses this simultaneous secrecy property, and hence the proof of Proposition 6.26 can be carried out also if the commitment scheme in used is the one of Construction 6.23 (see Exercise 14). We recall that this latter construction constitutes a commitment scheme if and only if such schemes exist at all (since Construction 6.23 is based on any one-way function and the existence of one-way functions is implied by the existence of commitment schemes).

Proposition 6.26 assumes the existence of a *nonuniformly* secure commitment scheme. The proof of the proposition makes essential use of the nonuniform security by incorporating instances on which the zero-knowledge property fails into circuits which contradict the security hypothesis. We stress that the sequence of "bad" instances is not necessarily constructible by efficient (uniform) machines. Put in other words, the zero-knowledge requirement has some nonuniform flavour. A *uniform analogue of zero-knowledge* would require only that it is infeasible to find instances on which a verifier gains knowledge (and not that such instances do not exist at all). Using a *uniformly* secure commitment scheme, Construction 6.25 can be shown to be *uniformly* zero-knowledge.

By itself, Construction 6.25 has little practical value, since it offers very moderate acceptance gap (between inputs inside and outside of the language). Yet, repeating the protocol, on common input $G = (V, E)$, for $k \cdot |E|$ times (and letting the verifier accept only if all iterations are accepting) yields an interactive proof for $G3C$ with error probability bounded

by e^{-k}, where $e \approx 2.718$ is the natural logarithm base. Namely, on common input $G \in G3C$ the verifier always accepts, whereas on common input $G \notin G3C$ the verifier accepts with probability bounded above by e^{-k} (no matter what the prover does). We stress that, by virtue of the Sequential Composition Lemma (Lemma 6.19), if these iterations are performed sequentially then the resulting (strong) interactive proof is zero-knowledge as well. Setting k to be any super-logarithmic function of $|G|$ (e.g., $k = |G|$), the error probability of the resulting interactive proof is negligible. We remark that it is unlikely that one can prove an analogous statement with respect to the interactive proof which results by performing these iteration in parallel. See Section 6.5.

An important property of Construction 6.25 is that the prescribed prover (i.e., P_{G3C}) can be implemented in probabilistic polynomial-time, provided that it is given as auxiliary input a 3-coloring of the common input graph. As we shall see, this property is essential to the applications of Construction 6.25 to the design of cryptographic protocols.

As admitted in the beginning of the current subsection, the choice of $G3C$ as a bootstrapping $\mathcal{NP}$-complete language is totally arbitrary. It is quite easy to design analogous zero-knowledge proofs for other popular $\mathcal{NP}$-complete languages. Such constructions will use the same underlying ideas as those presented in the *motivating discussion*.

6.4.3 The General Result and Some Applications

The theoretical and practical importance of a zero-knowledge proof for Graph 3-Coloring (e.g., Construction 6.25) follows from the fact that it can be applied to prove, in zero-knowledge, any statement having a short proof that can be efficiently verified. More precisely, a zero-knowledge proof system for a specific $\mathcal{NP}$-complete language (e.g., Construction 6.25) can be used to present zero-knowledge proof systems for every language in $\mathcal{NP}$.

Before presenting zero-knowledge proof systems for every language in $\mathcal{NP}$, let us recall some conventions and facts concerning $\mathcal{NP}$. We first recall that every language $L \in \mathcal{NP}$ is *characterized* by a binary relation R satisfying the following properties

- There exists a polynomial $p(\cdot)$ such that for every $(x, y) \in R$ it holds $|y| \leq p(|x|)$.
- There exists a polynomial-time algorithm for deciding membership in R.
- $L = \{x : \exists w \text{ s.t. } (x, w) \in R\}$.

Actually, each language in $\mathcal{NP}$ can be characterized by infinitely many such relations. Yet, for each $L \in \mathcal{NP}$ we fix and consider one characterizing relation, denoted R_L. Secondly, since $G3C$ is $\mathcal{NP}$-complete, we know that L is polynomial-time reducible (i.e., Karp-reducible) to $G3C$. Namely, there exists a polynomial-time computable function, f, such

that $x \in L$ if and only if $f(x) \in G3C$. Thirdly, we observe that the standard reduction of L to $G3C$, denoted f_L, has the following additional property:

> There exists a polynomial-time computable function, denoted g_L, such that for every $(x, w) \in R_L$ it holds that $g_L(w)$ is a 3-coloring of $f_L(x)$.

We stress that the above additional property is not required by the standard definition of a Karp-reduction. Yet, it can be easily verified that the standard reduction f_L (i.e., the composition of the generic reduction of L to SAT, the standard reductions of SAT to $3SAT$, and the standard reduction of $3SAT$ to $G3C$) does have such a corresponding g_L. (See Exercise 16.) Using these conventions, we are ready to "reduce" the construction of zero-knowledge proof for $\mathcal{NP}$ to a zero-knowledge proof system for $G3C$.

Construction 6.27 (A zero-knowledge proof for a language $L \in \mathcal{NP}$):

- Common Input: *A string x (supposedly in L);*
- Auxiliary Input to the Prover: *A* witness, *w, for the membership of $x \in L$ (i.e., a string w such that $(x, w) \in R_L$).*
- Local pre-computation: *Each party computes $G \stackrel{\text{def}}{=} f_L(x)$. The prover computes $\psi \stackrel{\text{def}}{=} g_L(w)$.*
- Invoking a zero-knowledge proof for $G3C$: *The parties invoke a zero-knowledge proof on common input G. The prover enters this proof with auxiliary input ψ.*

Proposition 6.28 *Suppose that the subprotocol used in the last step of Construction 6.27 is indeed an auxiliary input zero-knowledge proof for $G3C$. Then Construction 6.27 constitutes an auxiliary input zero-knowledge proof for L.*

Proof: The fact that Construction 6.27 constitutes an interactive proof for L is immediate from the validity of the reduction (and the fact that it uses an interactive proof for $G3C$). In first glance it seems that the zero-knowledge property of Construction 6.27 follows as immediately. There is however a minor issue that one should not ignore. The verifier in the zero-knowledge proof for $G3C$, invoked in Construction 6.27, possesses not only the common input graph G but also the original common input x which reduces to G. This extra information might have helped this verifier to extract knowledge in the $G3C$ interactive proof, if it were not the case that this proof system is zero-knowledge also with respect to auxiliary input. can be dealt with using auxiliary input to the verifier in Details follow.

Suppose we need to simulate the interaction of a machine V^* with the prover, on common input x. Without loss of generality we may assume that machine V^* invokes an interactive

machine V^{**} which interacts with the prover of the $G3C$ interactive proof, on common input $G = f_L(x)$ and having auxiliary input x. Using the hypothesis that the $G3C$ interactive proof is auxiliary input zero-knowledge, it follows that there exists a simulator M^{**} that on input (G, x) simulates the interaction of V^{**} with the $G3C$-prover (on common input G and verifier's auxiliary input x). Hence, the simulator for Construction 6.27, denoted M^*, operates as follows. On input x, the simulator M^* computes $G \stackrel{\text{def}}{=} f_L(x)$ and outputs $M^{**}(G, x)$. The proposition follows. ■

We remark that an alternative way of resolving the minor difficulty addressed above is to observe that the function f_L (i.e., the one induced by the standard reductions) can be inverted in polynomial-time (see Exercise 17). In any case, we immediately get

Theorem 6.29 *Suppose that there exists a commitment scheme satisfying the (nonuniform) secrecy and the unambiguity requirements. Then every language in $\mathcal{NP}$ has an auxiliary input zero-knowledge proof system. Furthermore, the prescribed prover in this system can be implemented in probabilistic polynomial-time, provided it gets the corresponding $\mathcal{NP}$-witness as auxiliary input.*

We remind the reader that the condition of the theorem is satisfied if (and only if) there exists (non-uniformly) one-way functions. See Theorem 3.29 (asserting that one-way functions imply pseudorandom generators), Proposition 6.24 (asserting that pseudorandom generators imply commitment schemes), and Exercise 12 (asserting that commitment schemes imply one-way functions).

An Example: Proving properties of secrets

A typical application of Theorem 6.29 is to enable one party to prove some property of its secrets without revealing the secrets. For concreteness, consider a party, denoted S, sending encrypted messages (over a public channel) to various parties, denoted $R_1, ..., R_t$, and wishing to prove to some other party, denoted V, that all the corresponding plaintext messages are identical. Further suppose that the messages are sent to the receivers (i.e., the R_i's) using a secure public-key encryption scheme, and let $E_i(\cdot)$ denote the (probabilistic) encryption employed when sending a message to R_i. Namely, to send message M_i to R_i, the sender uniformly chooses $r_i \in \{0,1\}^n$, computes the encryption $E_i(r_i, M_i)$, and transmits it over the public channel. In order to prove that $C_1 = E_1(r_1, M)$ and $C_2 = E_2(r_2, M)$ both encrypt the same message it suffices to reveal r_1, r_2 and M. However, doing so reveals the message M to the verifier. Instead, one can prove in zero-knowledge that there exists r_1, r_2 and M such that $C_1 = E_1(r_1, M)$ and $C_2 = E_2(r_2, M)$. The existence of such a zero-knowledge proof follows from Theorem 6.29 and the fact that the statement to be proven is of NP-type. Formally, we define a language

$$L \stackrel{\text{def}}{=} \{(C_1, C_2) : \exists r_1, r_2, M \text{ s.t. } C_1 = E_1(r_1, M) \text{ and } C_2 = E_2(r_2, M)\}$$

Clearly, the language L is in $\mathcal{NP}$, and hence Theorem 6.29 can be applied. Additional examples are presented in Exercise 18.

Zero-Knowledge for any language in IP

Interestingly, the result of Theorem 6.29 can be extended "to the maximum"; in the sense that under the same conditions every language having an interactive proof system also has a zero-knowledge proof system. Namely,

Theorem 6.30 *Suppose that there exists a commitment scheme satisfying the (nonuniform) secrecy and unambiguity requirements. Then every language in $\mathcal{IP}$ has a zero-knowledge proof system.*

We believe that this extension does not have much practical significance. Theorem 6.30 is proven by first converting the interactive proof for L into one in which the verifier uses only "public coins" (i.e., an Arthur-Merlin proof); see Chapter 8. Next, the verifier's coin tosses are forced to be almost unbiased by using a coin tossing protocols (see section ****???). Finally, the prover's replies are sent using a commitment scheme, At the end of the interaction the prover proves in zero-knowledge that the original verifier would have accepted the hidden transcript (this is an NP-statement).

6.4.4 Efficiency Considerations

When presenting zero-knowledge proof systems for every language in $\mathcal{NP}$, we made no attempt to present the most efficient construction possible. Our main concern was to present a proof which is as simple to explain as possible. However, once we know that zero-knowledge proofs for $\mathcal{NP}$ exist, it is natural to ask how efficient can they be.

In order to establish common grounds for comparing zero-knowledge proofs, we have to specify a desired measure of error probability (for these proofs). An instructive choice, used in the sequel, is to consider the complexity of zero-knowledge proofs with error probability 2^{-k}, where k is a parameter that may depend on the length of the common input. Another issue to bear in mind when comparing zero-knowledge proof is under what assumptions (if at all) are they valid. Throughout this entire subsection we stick to the assumption used so far (i.e., the existence of one-way functions).

Standard efficiency measures

Natural and standard efficiency measures to consider are

- The *communication complexity of the proof.* The most important communication measure is the *round complexity* (i.e., the number of message exchanges). The total number of bits exchanged in the interaction is also an important consideration.
- The *computational complexity of the proof.* Specifically the number of elementary steps taken by each of the parties.

Communication complexity seems more important than computational complexity, as long as the trade-off between them is "reasonable".

To demonstrate these measures we consider the zero-knowledge proof for $G3C$ presented in Construction 6.25. Recall that this proof system has very moderate acceptance gap, specifically $1/|E|$, on common input graph $G = (V, E)$. So Construction 6.25 has to be applied sequentially $k \cdot |E|$ in order to result in a zero-knowledge proof with error probability e^{-k}, where $e \approx 2.718$ is the natural logarithm base. Hence, the round complexity of the resulting zero-knowledge proof is $O(k \cdot |E|)$, the bit complexity is $O(k \cdot |E| \cdot |V|^2)$, and the computational complexity is $O(k \cdot |E| \cdot \mathrm{poly}(|V|))$, where the polynomial $\mathrm{poly}(\cdot)$ depends on the commitment scheme in use.

Much more efficient zero-knowledge proof systems may be custom-made for specific languages in $\mathcal{NP}$. Furthermore, even if one adopts the approach of reducing the construction of zero-knowledge proof systems for $\mathcal{NP}$ languages to the construction of a zero-knowledge proof system for a single $\mathcal{NP}$-complete language, efficiency improvements can be achieved. For example, using Exercise 15, one can present zero-knowledge proofs for the Hamiltonian Circuit Problem (again with error 2^{-k}) having round complexity $O(k)$, bit complexity $O(k \cdot |V|^{2+\epsilon})$, and computational complexity $O(k \cdot |V|^{2+O(\epsilon)})$, where $\epsilon > 0$ is a constant depending on the desired security of the commitment scheme (in Construction 6.25 and in Exercise 15 we chose $\epsilon = 1$). Note that complexities depending on the instance size are effected by reductions among problems, and hence a fair comparison is obtained by considering the complexities for the generic problem (i.e., Bounded Halting).

The round complexity of a protocol is a very important efficiency consideration and it is desirable to reduce it as much as possible. In particular, it is desirable to have zero-knowledge proofs with constant number of rounds and negligible error probability. This goal is pursued in Section 6.9.

Knowledge Tightness: a particular efficiency measure

The above efficiency measures are general in the sense that they are applicable to any protocol (independent on whether it is zero-knowledge or not). A particular measure of efficiency applicable to zero-knowledge protocols is their *knowledge tightness*. Intuitively, knowledge tightness is a refinement of zero-knowledge which is aimed at measuring the "actual security" of the proof system. Namely, how much harder does the verifier need to

work, when not interacting with the prover, in order to compute something which it can computes after interacting with the prover. Thus, knowledge tightness is the ratio between the (expected) running-time of the simulator and the running-time of the verifier in the real interaction simulated by the simulator. Note that the simulators presented so far, as well as all known simulator, operate by repeated random trials and hence an instructive measure of tightness should consider their expected running-time (assuming they never err (i.e., output the special $\perp$ symbol)) rather than the worst case.

Definition 6.31 (knowledge tightness): *Let $t : \mathbf{N} \mapsto \mathbf{N}$ be a function. We say that a zero-knowledge proof for language L has* knowledge tightness $t(\cdot)$ *if there exists a polynomial $p(\cdot)$ such that for every probabilistic polynomial-time verifier V^* there exists a simulator M^* (as in Definition 6.12) such that for all sufficiently long $x \in L$ we have*

$$\frac{\mathrm{Time}_{M^*}(x) - p(|x|)}{\mathrm{Time}_{V^*}(x)} \leq t(|x|)$$

where $\mathrm{Time}_{M^*}(x)$ denotes the expected running-time of M^* on input x, and $\mathrm{Time}_{V^*}(x)$ denotes the running time of V^* on common input x.

We assume a model of computation allowing one machine to invoke another machine at the cost of merely the running-time of the latter machine. The purpose of polynomial $p(\cdot)$, in the above definition, is to take care of generic overhead created by the simulation (this is important in case the verifier V^* is extremely fast). We remark that the definition of zero-knowledge does not guarantee that the knowledge tightness is polynomial. Yet, all known zero-knowledge proof, and more generally all zero-knowledge properties demonstrated using a single simulator with black-box access to V^*, have polynomial knowledge tightness. In particular, Construction 6.16 has knowledge tightness 2, whereas Construction 6.25 has knowledge tightness 3/2. We believe that knowledge tightness is a very important efficiency consideration and that it desirable to have it be a constant.

6.5 * Negative Results

In this section we review some negative results concerning zero-knowledge. These results can be viewed as evidence to the belief that some of the shortcomings of the results and constructions presented in previous sections are unavoidable. Most importantly, Theorem 6.29 asserts the existence of (computational) zero-knowledge proof systems for $\mathcal{NP}$, assuming that one-way functions exist. Two natural questions arise

1. *An unconditional result*: Can one prove the existence of (computational) zero-knowledge proof systems for $\mathcal{NP}$, without making any assumptions?

2. *Perfect zero-knowledge*: Can one present perfect zero-knowledge proof systems for $\mathcal{NP}$, even under some reasonable assumptions?

The answer to both question seems to be negative.

Another important question concerning zero-knowledge proofs is their preservation under parallel composition. We show that, *in general*, zero-knowledge is not preserved under parallel composition (i.e., there exists a pair of zero-knowledge protocols that when executed in parallel leak knowledge in a strong sense). Furthermore, we consider some natural proof systems, obtained via parallel composition of zero-knowledge proofs, and indicate that it is unlikely that the resulting composed proofs can be proven to be zero-knowledge.

6.5.1 Implausibility of an Unconditional "NP in ZK" Result

Recall that Theorem 6.30 asserts the existence of zero-knowledge proofs for any languages in $\mathcal{IP}$, *provided that nonuniform one-way functions exist*. In this subsection we consider the question of whether this sufficient condition is also necessary. The results, known to date, seem to provide some (yet, weak) indication in this direction. Specifically, the existence of zero-knowledge proof systems for languages out of $\mathcal{BPP}$ implies very weak forms of one-wayness. Also, the existence of zero-knowledge proof systems for languages which are hard to approximate, in some average case sense, implies the existence of one-way functions (but not of nonuniformly one-way functions). In the rest of this subsection we provide precise statements of the above results.

(1) $\mathcal{BPP} \subset \mathcal{CZK}$ implies weak forms of one-wayness

Definition 6.32 (collection of functions with one-way instances): *A collection of functions,* $\{f_i : D_i \mapsto \{0,1\}^*\}_{i \in \overline{I}}$, *is said to* have one-way instances *if there exists three probabilistic polynomial-time algorithms, I, D and F, so that the following two conditions hold*

1. easy to sample and compute: *as in Definition 2.11.*
2. some functions are hard to invert: *For every probabilistic polynomial-time algorithm, A', every polynomial $p(\cdot)$, and infinitely many i's*

$$\text{Prob}\left(A'(f_i(X_n), i) \in f_i^{-1} f_i(X_n)\right) < \frac{1}{p(n)}$$

where X_n is a random variable describing the output of algorithm D on input i.

Actually, since the hardness condition does not refer to the distribution induced by I, we may assume, without loss of generality, that $\overline{I} = \{0,1\}^*$ and algorithm I uniformly selects

a string (of length equal to the length of its input). Recall that collections of one-way functions (as defined in Definition 2.11) requires hardness to invert of all but a negligible measure of the functions f_i (where the probability measure is induced by algorithm I).

Theorem 6.33 *If there exist zero-knowledge proofs for languages outside of $\mathcal{BPP}$ then there exist collections of functions with one-way instances.*

We remark that the mere assumption that $\mathcal{BPP} \subset \mathcal{IP}$ is not known to imply any form of one-wayness. The existence of a language in $\mathcal{NP}$ which is not in $\mathcal{BPP}$ implies the existence of a function which is easy to compute but hard to invert in the worst-case (see Section 2.1). The latter consequence seems to be a much weaker form of one-wayness.

(2) zero-knowledge proofs for "hard" languages yield one-way functions

Our notion of hard languages is the following

Definition 6.34 *We say that a language L is* hard to approximate *if there exists a probabilistic polynomial-time algorithm S such that for every probabilistic polynomial-time algorithm A, every polynomial $p(\cdot)$, and infinitely many n's*

$$\mathrm{Prob}(A(X_n) = \chi_L(X_n)) < \frac{1}{2} + \frac{1}{p(n)}$$

where $X_n \stackrel{\mathrm{def}}{=} S(1^n)$, and χ_L is the characteristic function of the language L (i.e., $\chi_L(x) = 1$ if $x \in L$ and $\chi_L(x) = 0$ otherwise).

Theorem 6.35 *If there exist zero-knowledge proofs for languages that are hard to approximate then there exist one-way functions.*

We remark that the mere existence of languages that are hard to approximate (even in a stronger sense by which the approximater must fail on all sufficiently large n's) is not known to imply the existence of one-way functions (see Section 2.1).

6.5.2 Implausibility of Perfect Zero-Knowledge proofs for all of NP

A theorem bounding the class of languages possessing *perfect* zero-knowledge proof systems follows. We start with some background (for more details see Section `[missing(eff-ip.sec)]`). By $\mathcal{AM}$ we denote the class of languages having an interactive proof which proceeds as follows. First the verifier sends a random string to the prover, next the prover answers with

some string, and finally the verifier decided whether to accept or reject based on a deterministic computation (depending on the common input and the above two strings). The class $\mathcal{AM}$ seems to be a randomized counterpart of $\mathcal{NP}$, and it is believed that $co\mathcal{NP}$ is not contained in $\mathcal{AM}$. Additional support to this belief is given by the fact that $co\mathcal{NP} \subseteq \mathcal{AM}$ implies the collapse of the Polynomial-Time Hierarchy. In any case it is known that

Theorem 6.36 *The class of languages possessing perfect zero-knowledge proof systems is contained in the class $co\mathcal{AM}$. (In fact, these languages are also in $\mathcal{AM}$.)*

The theorem remains valid under several relaxations of perfect zero-knowledge (e.g., allowing the simulator to run in expected polynomial-time, etc.). Hence, if some $\mathcal{NP}$-complete language has a perfect zero-knowledge proof system then $co\mathcal{NP} \subseteq \mathcal{AM}$, which is unlikely.

We stress that Theorem 6.36 *does not* apply to perfect zero-knowledge arguments, defined and discussed in Section 6.8. Hence, there is no conflict between Theorem 6.36 and the fact that, under some reasonable complexity assumptions, perfect zero-knowledge arguments do exist for every language in $\mathcal{NP}$.

6.5.3 Zero-Knowledge and Parallel Composition

We discuss two negative results of very different conceptual standing. The first result asserts the failure of the general "Parallel Composition Conjecture", but says nothing about specific natural candidates. The second result refers to a class of interactive proofs, which contains several interesting and natural examples, and assert that the members of this class cannot be proven zero-knowledge using a general paradigm (know by the name "black box simulation"). We mention that it is hard to conceive an alternative way of demonstrating the zero-knowledge property of protocols (rather than by following this paradigm).

(1) Failure of the Parallel Composition Conjecture

For some time, after zero-knowledge proofs were first introduced, several researchers insisted that the following must be true

Parallel Composition Conjecture: *Let P_1 and P_2 be two zero-knowledge provers. Then the prover resulting by running both of them in parallel is also zero-knowledge.*

Some researchers even considered the failure to prove the Parallel Composition Conjecture as a sign of incompetence. However, the Parallel Composition Conjecture is just wrong.

Proposition 6.37 *There exists two provers, P_1 and P_2, such that each is zero-knowledge, and yet the prover resulting by running both of them in parallel yields knowledge (e.g., a cheating verifier may extract from this prover a solution to a problem that is not solvable in polynomial-time). Furthermore, the above holds even if the zero-knowledge property of each of the P_i's can be demonstrated using a simulator which uses the verifier as a black-box (see below).*

We remark that these provers can be incorporated into a single prover that randomly selects which of the two programs to execute. Alternatively, the choice may be determined by the verifier.

Proof idea: Consider a prover, denoted P_1, that send "knowledge" to the verifier if and only if the verifier can answer some randomly chosen *hard question* (i.e., we stress that the question is chosen by P_1). Answers to the hard questions look pseudorandom, yet P_1 (which is not computationally bounded) can verify their correctness. Now, consider a second prover, denoted P_2, that answers these hard questions. Each of these provers (by itself) is zero-knowledge: P_1 is zero-knowledge since it is unlikely that any probabilistic polynomial-time verifier can answer its questions; whereas P_2 is zero-knowledge since its answers can be simulated by random strings. Yet, once played in parallel, a cheating verifier can answer the question of P_1 by sending it to P_2, and using this answer gain knowledge from P_1. To turn this idea into a proof we need to implement a hard problem with the above properties. ■

The above proposition refutes the Parallel Composition Conjecture by means of exponential time provers. Assuming the existence of one-way functions the Parallel Composition Conjecture can be refuted also for probabilistic polynomial-time provers (with auxiliary inputs). For example, consider the following two provers P_1 and P_2, which make use of proofs of knowledge (see Section 6.7). Let C be a bit commitment scheme (which we know to exist provided that one-way functions exist). On common-input $C(1^n, \sigma)$, where $\sigma \in \{0,1\}$, prover P_1 proves to the verifier, in zero-knowledge, that it knows σ. (To this end the prover is give as auxiliary input the coins used in the commitment.) On input $C(1^n, \sigma)$, prover P_2 asks the verifier to prove that it knows σ and if P_2 is convinced then it sends σ to the verifier. This verifier employs the same system of proofs of knowledge used by the program P_1. Clearly, each prover is zero-knowledge and yet their parallel composition is not. Similarly, using stronger intractability assumptions, one can refute the Parallel Composition Conjecture also with respect to perfect zero-knowledge (rather than with respect to computational zero-knowledge).

(2) Problems with "natural" candidates

By definition, to show that a prover is zero-knowledge one has to present, for each prospective verifier V^*, a corresponding simulator M^* (which simulates the interaction of V^* with

the prover). However, all known demonstrations of zero-knowledge proceed by presenting one "universal" simulator which uses any prospective verifier V^* as a black-box. In fact, these demonstrations use as black-box (or oracle) the "next message" function determined by the verifier program (i.e., V^*), its auxiliary-input and its random-input. (This property of the simulators is implicit in our constructions of the simulators in previous sections.) We remark that it is hard to conceive an alternative way of demonstrating the zero-knowledge property.

Definition 6.38 (black-box zero-knowledge):

- next message function: *Let B be an interactive turing machine, and x, z, r be strings representing a common-input, auxiliary-input, and random-input, respectively. Consider the function $B_{x,z,r}(\cdot)$ describing the messages sent by machine B such that $B_{x,z,r}(\overline{m})$ denotes the message sent by B on common-input x, auxiliary-input z, random-input r, and sequence of incoming messages $\overline{m}$. For simplicity, we assume that the output of B appears as its last message.*
- black-box simulator: *We say that a probabilistic polynomial-time oracle machine M is a* black-box simulator for the prover P and the language L *if for every polynomial-time interactive machine B, every probabilistic polynomial-time oracle machine D, every polynomial $p(\cdot)$, all sufficiently large $x \in L$, and every $z, r \in \{0,1\}^*$:*

$$|\text{Prob}\left(D^{B_{x,z,r}}(\langle P, B_r(z)\rangle(x))=1\right) - \text{Prob}\left(D^{B_{x,z,r}}(M^{B_{x,z,r}}(x))=1\right)| < \frac{1}{p(|x|)}$$

  where $B_r(z)$ denotes the interaction of machine B with auxiliary-input z and random-input r.
- *We say that P is* black-box zero knowledge *if it has a black-box simulator.*

Essentially, the definition says that a black-box simulator mimics the interaction of prover P with any polynomial-time verifier B, relative to any auxiliary-input (i.e., z) that B may get and any random-input (i.e., r) that B may choose. The simulator does so (efficiently), merely by using oracle calls to $B_{x,z,r}$ (which specifies the next message that B sends on input x, auxiliary-input z, and random-input r). The simulation is indistinguishable from the true interaction, even if the distinguishing algorithm (i.e., D) is given access to the oracle $B_{x,z,r}$. An equivalent formulation is presented in Exercise 23. Clearly, if P is black-box zero-knowledge then it is zero-knowledge with respect to auxiliary input (and has polynomially bounded knowledge tightness (see Definition 6.31)).

Theorem 6.39 *Suppose that (P,V) is an interactive proof system, with negligible error probability, for the language L. Further suppose that (P,V) has the following properties*

- constant round: *There exists an integer k such that for every $x \in L$, on input x the prover P sends at most k messages.*
- public coins: *The messages sent by the verifier V are predetermined consecutive segments of its random tape.*
- black-box zero-knowledge: *The prover P has a black-box simulator (over the language L).*

Then $L \in \mathcal{BPP}$.

We remark that both Construction 6.16 (zero-knowledge proof for Graph Isomorphism) and Construction 6.25 (zero-knowledge proof for Graph Coloring) are constant round, use public coins and are black-box zero-knowledge (for the corresponding languages). However, they *do not* have negligible error probability. Yet, repeating each of these constructions polynomially many times *in parallel* yields an interactive proof, with negligible error probability, for the corresponding language. Clearly the resulting proof system are constant round and use public coins. Hence, unless the corresponding languages are in $\mathcal{BPP}$, these parallelized proof systems *are not* black-box zero-knowledge.

Theorem 6.39 is sometimes interpreted as pointing to an inherent limitation of interactive proofs with public coins (also known as *Arthur Merlin* games; see Section `[missing(eff-ip.sec)]`). Such proofs cannot be both round-efficient (i.e., have constant number of rounds and negligible error) and black-box zero-knowledge (unless they are trivially so, i.e., the language is in $\mathcal{BPP}$). In other words, *when constructing round-efficient zero-knowledge proof systems* (for languages not in $\mathcal{BPP}$), *one is advised to use "private coins"* (i.e., to let the verifier send messages depending upon, but not revealing its coin tosses).

6.6 * Witness Indistinguishability and Hiding

In light of the non-closure of zero-knowledge under parallel composition, see Subsection 6.5.3, alternative "privacy" criteria that are preserved under parallel composition are of practical and theoretical importance. Two notions, called witness indistinguishability and witness hiding, which refer to the "privacy" of interactive proof systems (of languages in $\mathcal{NP}$), are presented in this section. Both notions seem weaker than zero-knowledge, yet they suffice for some specific applications.

6.6.1 Definitions

In this section we confine ourself to languages in $\mathcal{NP}$. Recall that a *witness relation* for a language $L \in \mathcal{NP}$ is a binary relation R_L that is polynomially-bounded (i.e., $(x,y) \in R_L$

implies $|y| \leq \text{poly}(|x|)$), polynomial-time recognizable, and characterizes L by

$$L = \{x : \exists y \text{ s.t. } (x,y) \in R_L\}$$

Witness indistinguishability

Loosely speaking, an interactive proof for a language $L \in \mathcal{NP}$ is *witness independent* (resp., *witness indistinguishable*) if the verifier's view of the interaction with the prover is statistically independent (resp., "computationally independent") of the auxiliary input of the prover. Actually, we will relax the requirement so that it applies only to the case in which the auxiliary input constitutes an NP-witness to the common input; namely, let R_L be the witness relation of the language L and suppose that $x \in L$, then we consider only auxiliary inputs in $R_L(x) \stackrel{\text{def}}{=} \{y; (x,y) \in R_L\}$. By saying that the view is computational independent of the witness we mean that for every two choices of auxiliary inputs the resulting views are computationally indistinguishable. In the actual definition we combine notations and conventions from Definitions 6.13 and 6.18.

Definition 6.40 (witness indistinguishability / independence): *Let (P,V), $L \in \mathcal{NP}$ and V^* be as in Definition 6.18, and let R_L be a fixed witness relation for the language L. We denote by $\text{view}^{P(y)}_{V^*(z)}(x)$ a random variable describing the contents of the random-tape of V^* and the messages V^* receives from P during a joint computation on common input x, when P has auxiliary input y and V^* has auxiliary input z. We say that (P,V) is* witness indistinguishable *for R_L if for every probabilistic polynomial-time interactive machine V^*, and every two sequences $W^1 = \{w^1_x\}_{x \in L}$ and $W^2 = \{w^2_x\}_{x \in L}$, so that $w^1_x, w^2_x \in R_L(x)$, the following two ensembles are computationally indistinguishable*

- $\{x, \text{view}^{P(w^1_x)}_{V^*(z)}(x)\}_{x \in L, z \in \{0,1\}^*}$
- $\{x, \text{view}^{P(w^2_x)}_{V^*(z)}(x)\}_{x \in L, z \in \{0,1\}^*}$

Namely, for every probabilistic polynomial-time algorithm, D, every polynomial $p(\cdot)$, all sufficiently long $x \in L$, and all $z \in \{0,1\}^*$, it holds that

$$|\text{Prob}(D(x, \text{view}^{P(w^1_x)}_{V^*(z)}(x)) = 1) - \text{Prob}(D(x, \text{view}^{P(w^2_x)}_{V^*(z)}(x)) = 1)| < \frac{1}{p(|x|)}$$

We say that (P,V) is witness independent *if the above ensembles are identically distributed. Namely, for every $x \in L$ every $w^1_x, w^2_x \in R(x)$ and $z \in \{0,1\}^*$, the random variables $\text{view}^{P(w^1_x)}_{V^*(z)}(x)$ and $\text{view}^{P(w^2_x)}_{V^*(z)}(x)$ are identically distributed.*

A few remarks are in place. First, one may observe that any proof system in which the prover ignores its auxiliary-input is trivially witness independent. In particular, exponential-time provers may, without loss of generality, ignore their auxiliary-input (without any decrease in the probability that they convince the verifier). Yet, probabilistic polynomial-time provers can not afford to ignore their auxiliary input (since otherwise they become useless). Hence, for probabilistic polynomial-time provers for languages outside $\mathcal{BPP}$, witness indistinguishability is non-trivial. Secondly, one can easily show that any zero-knowledge proof system for a language in $\mathcal{NP}$ is witness indistinguishable (since the view corresponding to each witness can be approximated by the same simulator). Likewise, perfect zero-knowledge proofs are witness independent. Finally, it is relatively easy to see that witness indistinguishability and witness independence are preserved under sequential composition. In the next subsection we show that they are also preserved under parallel composition.

Witness hiding

We now turn to the notion of witness hiding. Intuitively, a proof system for a language in $\mathcal{NP}$ is witness hiding if after interacting with the prover it is still infeasible for the verifier to find an NP witness for the common input. Clearly, such a requirement can hold only if it is infeasible to find witnesses from scratch. Since, each $\mathcal{NP}$ language has instances for which witness finding is easy, we must consider the task of witness finding for specially selected hard instances. This leads to the following definitions.

Definition 6.41 (distribution of hard instances): *Let $L \in \mathcal{NP}$ and R_L be a witness relation for L. Let $X \stackrel{\text{def}}{=} \{X_n\}_{n\in\mathbf{N}}$ be a probability ensemble so that X_n assign non-zero probability mass only to strings in $L \cap \{0,1\}^n$. We say that X is* hard for *R_L if for every probabilistic polynomial-time* (witness finding) *algorithm F, every polynomial $p(\cdot)$, all sufficiently large n's and all $z \in \{0,1\}^{\text{poly}(n)}$*

$$\text{Prob}(F(X_n, z) \in R_L(X_n)) < \frac{1}{p(n)}$$

Definition 6.42 (witness hiding): *Let (P,V), $L \in \mathcal{NP}$, and R_L be as in the above definitions.*

- *Let $X = \{X_n\}_{n\in\mathbf{N}}$ be a hard instance ensemble for R_L. We say that (P,V) is* witness hiding for *the relation R_L* under *the instance ensemble X if for every probabilistic polynomial-time machine V^*, every polynomial $p(\cdot)$ and all sufficiently large n's, and all $z \in \{0,1\}^*$*

  $$\text{Prob}(\langle P(Y_n), V^*(z)\rangle(X_n) \in R_L(X_n)) < \frac{1}{p(n)}$$

  where Y_n is arbitrarily distributed over $R_L(X_n)$.

- *We say that* (P,V) *is* universal witness hiding for the relation R_L *if the proof system* (P,V) *is witness hiding for* R_L *under every ensemble of hard instances, for* R_L*, that is efficiently constructible (see Definition 3.5)*

We remark that the relation between the two privacy criteria (i.e., witness indistinguishable and witness hiding) is not obvious. Yet, zero-knowledge proofs (for $\mathcal{NP}$) are also (universal) witness hiding (for any corresponding witness relation). We remark that witness indistinguishability and witness hiding, similarly to zero-knowledge, are properties of the prover (and more generally of a any interactive machine).

6.6.2 Parallel Composition

In contrary to zero-knowledge proof systems, witness indistinguishable proofs offer some robustness under parallel composition. Specifically, parallel composition of witness indistinguishable proof systems results in a witness indistinguishable system, *provided that the original prover is probabilistic polynomial-time.*

Lemma 6.43 (Parallel Composition Lemma): *Let* $L \in \mathcal{NP}$*, and* R_L *be as in Definition 6.40, and suppose that* P *is probabilistic polynomial-time, and* (P,V) *is* witness indistinguishable (resp., witness independent) *for* R_L*. Let* $Q(\cdot)$ *be a polynomial, and* P_Q *denote a program that on common-input* $x_1,...,x_{Q(n)} \in \{0,1\}^n$ *and auxiliary-input* $w_1,...,w_{Q(n)} \in \{0,1\}^*$*, invokes* P *in parallel* $Q(n)$ *times, so that in the* i^{th} *copy* P *is invoked on common-input* x_i *and auxiliary-input* w_i*. Then,* P_Q *is* witness indistinguishable (resp., witness independent) *for*

$$R_L^Q \stackrel{\text{def}}{=} \{(\overline{x},\overline{w}) : \forall i\ (x_i,w_i) \in R_L\}$$

where $\overline{x} = (x_1,...,x_m)$*, and* $\overline{w} = (w_1,...,w_m)$*, so that* $m = Q(n)$ *and* $|x_i| = n$ *for each* i*.*

Proof Sketch: Both the computational and information theoretic versions follow by a hybrid argument. We concentrate on the computational version. To avoid cumbersome notation we consider a generic n for which the claim of the lemma fails. (By contradiction there must be infinitely many such n's and a precise argument will actually handle all these n's together.) Namely, suppose that by using a verifier program V_Q^*, it is feasible to distinguish the witnesses $\overline{w}^1 = (w_1^1,...,w_m^1)$ and $\overline{w}^2 = (w_1^2,...,w_m^2)$, used by P_Q, in an interaction on common-input $\overline{x} \in L^m$. Then, for some i, the program V_Q^* distinguishes also the hybrid witnesses $\overline{h}^{(i)} = (w_1^1,...,w_i^1,w_{i+1}^2,...,w_m^2)$ and $\overline{h}^{(i+1)} = (w_1^1,...,w_{i+1}^1,w_{i+2}^2,...,w_m^2)$. Rewrite $\overline{h}^{(i)} = (w_1,...,w_i,w_{i+1}^2,w_{i+2},...,w_m)$ and $\overline{h}^{(i+1)} = (w_1,...,w_i,w_{i+1}^1,w_{i+2},...,w_m)$. We derive a contradiction by constructing a verifier V^* that distinguishes (the witnesses used by P in) interactions with the original prover P. Details follow.

The program V^* incorporates the programs P and V_Q^* and proceeds by interacting with the prover P in parallel to simulating $m-1$ other interactions with P. The real interaction with P is viewed as the $i+1^{\text{st}}$ copy in an interaction of V_Q^*, whereas the simulated interactions are associated with the other copies. Specifically, in addition to the common-input x, machine V^* gets the appropriate i and the sequences $\overline{x}$, $\overline{h}^{(i)}$ and $\overline{h}^{(i+1)}$ as part of its auxiliary input. For each $j \neq i+1$, machine V^* will use x_j as common-input and w_j as the auxiliary-input to the j^{th} copy of P. Machine V^* invokes V_Q^* on common input $\overline{x}$ and provides it with an interface to a virtual interaction with P_Q. The $i+1^{\text{st}}$ component of a message $\overline{\alpha} = (\alpha_1, ..., \alpha_m)$ sent by V_Q^* is forwarded to the prover P and all other components are kept for the simulation of the other copies. When P answers with a message β, machine V^* computes the answers of the other copies of P (by feeding the program P with the corresponding auxiliary-input and the corresponding sequence of incoming messages). It follows, that V^* can distinguish the case P uses the witness w_{i+1}^1 from the case P uses w_{i+1}^2. ■

6.6.3 Constructions

In this subsection we present constructions of witness indistinguishable and witness hiding proof systems.

Constructions of witness indistinguishable proofs

Using the Parallel Composition Lemma and the observation that zero-knowledge proofs are witness indistinguishable we derive the following

Theorem 6.44 *Assuming the existence of (nonuniformly) one-way functions, every language in $\mathcal{NP}$ has a* constant-round *witness indistinguishable proof system with* negligible error probability. *In fact, the error probability can be made exponentially small.*

We remark that no such result is known for zero-knowledge proof system. Namely, the known proof systems for $\mathcal{NP}$ are either

- not constant-round (e.g., Construction 6.27); or
- have non-negligible error probability (e.g., Construction 6.25); or
- require stronger intractability assumptions (see Subsection 6.9.1); or
- are only computationally sound (see Subsection 6.9.2).

Similarly, we can derive a constant-round witness independent proof system, with exponentially small error probability, for Graph Isomorphism. (Again, no analogous result is known for perfect zero-knowledge proofs.)

Constructions of witness hiding proofs

Witness indistinguishable proof systems are not necessarily witness hiding. For example, any language with unique witnesses has a proof system which yields the unique witness, and yet is trivially witness independent. On the other hand, for some relations, witness indistinguishability implies witness hiding. For example

Proposition 6.45 *Let $\{(f_i^0, f_i^1) : i \in \overline{I}\}$ be a collection of (nonuniform) clawfree functions, and let*

$$R \stackrel{\text{def}}{=} \{(x, w) : w = (\sigma, r) \wedge x = (i, x') \wedge x' = f_i^\sigma(r)\}$$

Then if a machine P is witness indistinguishable for R then it is also witness hiding for R under the distribution generated by setting $i = I(1^n)$ and $x' = f_i^0(D(0, i))$, where I and D are as in Definition 2.13.

By a collection of nonuniform clawfree functions we mean that even nonuniform families of circuits $\{C_n\}$ fail to form claws on input distribution $I(1^n)$, except with negligible probability. We remark that the above proposition does not relate to the purpose of interacting with P (e.g., whether P is proving membership in a language, knowledge of a witness, and so on). The proposition is proven by contradiction. Details follow.

Suppose that an interactive machine V^* finds witnesses after interacting with P. By the witness indistinguishability of P it follows that V^* is performing as well regardless on whether the witness is of the form $(0, \cdot)$ or $(1, \cdot)$. Combining the programs V^* and P with algorithm D we derive a claw forming algorithm (and hence contradiction). Specifically, the claw-forming algorithm, on input $i \in \overline{I}$, uniformly selects $\sigma \in \{0, 1\}$, randomly generates $r = D(\sigma, i)$, computes $x = (i, f_i^\sigma(r))$, and simulates an interaction of V^* with P on common-input x and auxiliary-input (σ, r) to P. If machine V^* outputs a witness $w \in R(x)$ then, with probability approximately $\frac{1}{2}$, we have $w = (1 - \sigma, r')$ and a claw is formed (since $f_i^\sigma(r) = f_i^{1-\sigma}(r')$).□

Furthermore, every NP relation can be "slightly modified" so that, for the modified relation, witness indistinguishability implies witness hiding. Given a relation R, the modified relation, denoted R_2, is defined by

$$R_2 \stackrel{\text{def}}{=} \{((x_1, x_2), w) : |x_1| = |x_2| \wedge \exists i \text{ s.t. } (x_i, w) \in R\}$$

Namely, w is a witness under R_2 for the instance (x_1, x_2) if and only if w is a witness under R for either x_1 or x_2.

Proposition 6.46 *Let R and R_2 be as above. If a machine P is witness indistinguishable for R_2 then it is also witness hiding for R_2 under every distribution of hard instances induced (see below) by an efficient algorithm that randomly selects pairs in R.*

Let S be a probabilistic polynomial-time algorithm that on input 1^n outputs $(x,w) \in R$ so that $|x| = n$. Let X_n denotes the distribution induced on the first element in the output of $S(1^n)$. The proposition asserts that if P is witness indistinguishable and $\{X_n\}_{n\in\mathbb{N}}$ an ensemble of hard instances for R then P is witness hiding under the ensemble $\{\overline{X}_n\}_{n\in\mathbb{N}}$ where $\overline{X}_n$ consists of two independent copies of X_n. This assertion is proven by contradiction. Suppose that an interactive machine V^* finds witnesses after interacting with P. By the witness indistinguishability of P it follows that V^* is performing as well regardless on whether the witness w for (x_1,x_2) satisfies either $(x_1,w) \in R$ or $(x_2,w) \in R$. Combining the programs V^* and P with algorithm S we derive a algorithm, denoted F^*, that finds witnesses for R (under the distribution X_n). On input $x \in L$, algorithm F^* generates at random $(x',w') = S(1^{|x|})$ and sets $\overline{x} = (x,x')$ with probability $\frac{1}{2}$ and $\overline{x} = (x',x)$ otherwise. Algorithm F^* simulates an interaction of V^* with P on common-input $\overline{x}$ and auxiliary input w' to P, and when V^* outputs a witness w algorithm F^* checks whether $(x,w) \in R$. The reader can easily verifier that algorithm F^* performs well under the instance ensemble $\{X_n\}$, hence contradicting the hypothesis that X_n is hard for R. □

6.6.4 Applications

Applications for the notions presented in this section are scattered in various places in the book. In particular, witness-indistinguishable proof systems are used in the construction of constant-round arguments for $\mathcal{NP}$ (see Subsection 6.9.2), witness independent proof systems are used in the zero-knowledge proof for Graph Non-Isomorphism (see Section 6.7), and witness hiding proof systems are used for the efficient identification scheme based on factoring (in Section 6.7).

6.7 * Proofs of Knowledge

This section addresses the concept of "proofs of knowledge". Loosely speaking, these are proofs in which the prover asserts "knowledge" of some object (e.g., a 3-coloring of a graph) and not merely its existence (e.g., the existence of a 3-coloring of the graph, which in turn imply that the graph is in the language $G3C$). But what is meant by saying that a machine knows something? Indeed the main thrust of this section is in addressing this question. Before doing so we point out that "proofs of knowledge", and in particular zero-knowledge "proofs of knowledge", have many applications to the design of cryptographic schemes and cryptographic protocols. Some of these applications are discussed in a special subsection. Of special interest is the application to identification schemes, which is discussed in a separate subsection.

6.7.1 Definition

We start with a motivating discussion.

> What does it mean to say that a machine knows something? Any standard dictionary suggests several meanings to the verb *know* and most meanings are phrased with reference to "awareness". We, however, must look for a behavioristic interpretation of the verb. Indeed, it is reasonable to link knowledge with ability to do something, be it at the least the ability to write down whatever one knows. Hence, we will say that a machine knows a string α if it *can* output the string α. This seems as total nonsense. A machine has a well defined output: either the output equals α or it does not. So what can be meant by saying that a machine can do something. Loosely speaking, it means that the machine can be modified so that it does whatever is claimed. More precisely, it means that there exists an *efficient* machine which, using the original machine as oracle, outputs whatever is claimed.

So far for defining the "knowledge of machines". Yet, whatever a machine knows or does not know is "its own business". What can be of interest to the outside is the question of what can be deduced about the knowledge of a machine after interacting with it. Hence, we are interested in proofs of knowledge (rather than in mere knowledge).

For sake of simplicity let us consider a concrete question: how can a machine prove that it knows a 3-coloring of a graph? An obvious way is just to send the 3-coloring to the verifier. Yet, we claim that applying Construction 6.25 (i.e., the zero-knowledge proof system for $G3C$) sufficiently many times results in an alternative way of proving knowledge of a 3-coloring of the graph. Loosely speaking, we say that an interactive machine, V, constitutes a *verifier for knowledge* of 3-coloring if the probability that the verifier is convinced by a machine P to accept the graph G is inversely proportional to the difficulty of extracting a 3-coloring of G when using machine P as a "black box". Namely, the extraction of the 3-coloring is done by an oracle machine, called an *extractor*, that is given access to a function specifying the messages sent by P (in response to particular messages that P receives). The (expected) running time of the extractor, on input G and access to an oracle specifying P's messages, is inversely related (by a factor polynomial in $|G|$) to the probability that P convinces V to accept G. In case P always convinces V to accept G, the extractor runs in expected polynomial-time. The same holds in case P convinces V to accept with non-negligible probability. We stress that the latter special cases do not suffice for a satisfactory definition.

Preliminaries

Let $R \subseteq \{0,1\}^* \times \{0,1\}^*$ be a binary relation. Then $R(x) \stackrel{\text{def}}{=} \{s : (x,s) \in R\}$ and $L_R \stackrel{\text{def}}{=} \{x : \exists s \text{ s.t. } (x,s) \in R\}$. If $(x,s) \in R$ then we call s a *solution* for x. We say that R is

polynomially bounded if there exists a polynomial p such that $|s| \leq p(|x|)$ for all $(x,s) \in R$. We say that R is an *NP relation* if R is polynomially bounded and, in addition, there exists a polynomial-time algorithm for deciding membership in R (i.e., $L_R \in \mathcal{NP}$). In the sequel, we confine ourselves to polynomially bounded relations.

We wish to be able to consider in a uniform manner all potential provers, without making distinction based on their running-time, internal structure, etc. Yet, we observe that these interactive machine can be given an auxiliary-input which enables them to "know" and to prove more. Likewise, they may be luck to select a random-input which enables more than another. Hence, statements concerning the knowledge of the prover refer not only to the prover's program but also to the specific auxiliary and random inputs it has. Hence, we fix an interactive machine and all inputs (i.e., the common-input, the auxiliary-input, and the random-input) to this machine, and consider both the corresponding accepting probability (of the verifier) and the usage of this (prover+inputs) template as an oracle to a "knowledge extractor". This motivates the following definition.

Definition 6.47 (message specification function)*: Denote by $P_{x,y,r}(\overline{m})$ the message sent by machine P on common-input x, auxiliary-input y, and random input r, after receiving messages $\overline{m}$. The function $P_{x,y,r}$ is called the* message specification function *of machine P with common-input x, auxiliary-input y, and random input r.*

An oracle machine with access to the function $P_{x,y,r}$ will represent the knowledge of machine P on common-input x, auxiliary-input y, and random input r. This oracle machine, called the knowledge extractor, will try to find a solution to x (i.e., an $s \in R(x)$). The running time of the extractor is inversely related to the corresponding accepting probability (of the verifier).

Knowledge verifiers

Now that all the machinery is ready, we present the definition of a system for proofs of knowledge. Actually, the definition presented below is a generalization (to be motivated by the subsequent applications). At first reading, the reader may set the function κ to be identically zero.

Definition 6.48 (System of proofs of knowledge): *Let R be a binary relation, and $\kappa : \mathbb{N} \to [0,1]$. We say that an interactive function V is a* knowledge verifier for the relation R *with* knowledge error κ *if the following two conditions hold.*

- Non-triviality: *There exists an interactive machine P so that for every $(x,y) \in R$ all possible interactions of V with P on common-input x and auxiliary-input y are accepting.*

- Validity (with error κ): *There exists a polynomial $q(\cdot)$ and a probabilistic oracle machine K such that for every interactive function P, every $x \in L_R$ and every $y, r \in \{0,1\}^*$, machine K satisfies the following condition:*

  > *Denote by $p(x)$ the probability that the interactive machine V accepts, on input x, when interacting with the prover specified by $P_{x,y,r}$. Then if $p(x) > \kappa(|x|)$ then, on input x and access to oracle $P_{x,y,r}$, machine K outputs a solution $s \in R(x)$ within an expected number of steps bounded by*
  >
  > $$\frac{q(|x|)}{p(x) - \kappa(|x|)} .$$

  The oracle machine K is called a universal knowledge extractor.

When $\kappa(\cdot)$ is identically zero, we just say that V is a knowledge verifier for the relation *R. An interactive pair (P,V) so that V is a knowledge verifier for a relation R and P is a machine satisfying the non-triviality condition (with respect to V and R) is called a* system for proofs of knowledge *for the relation R.*

6.7.2 Observations

The zero-knowledge proof systems for Graph Isomorphism (i.e., Construction 6.16) and for Graph 3-Coloring (i.e., Construction 6.25) are in fact proofs of knowledge (with some knowledge error) for the corresponding languages. Specifically, Construction 6.16 is a proof of knowledge of an isomorphism with knowledge error $\frac{1}{2}$, whereas Construction 6.25 is a proof of knowledge of a 3-coloring with knowledge error $1 - \frac{1}{|E|}$ (on common input $G = (V,E)$). By iterating each construction sufficiently many times we can get the knowledge error to be exponentially small. (The proofs of all these claims are left as an exercise.) In fact, we get a proof of knowledge with zero error, since

Proposition 6.49 *Let R be an NP relation, and $q(\cdot)$ be a polynomial such that $(x,y) \in R$ implies $|y| \leq q(|x|)$. Suppose that (P,V) is a system for proofs of knowledge, for the relation R, with knowledge error $\kappa(n) \stackrel{\text{def}}{=} 2^{-q(n)}$. Then (P,V) is a system for proofs of knowledge for the relation R (with zero knowledge error).*

Proof Sketch: Given a knowledge extractor, K, substantiating the hypothesis, we construct a new knowledge extractor which runs K in parallel to conducting an exhaustive search for a solution. Let $p(x)$ be as in Definition 6.48. To evaluate the performance of the new extractor consider two cases.

Case 1: $p(x) \geq 2 \cdot \kappa(|x|)$. In this case, we use the fact

$$\frac{1}{p(x) - \kappa(|x|)} \leq \frac{2}{p(x)}$$

Case 2: $p(x) \leq 2 \cdot \kappa(|x|)$. In this case, we use the fact that exhaustive search of a solution boils down to $2^{q(|x|)}$ trials, whereas $\frac{1}{p(x)} \geq \frac{1}{2} \cdot 2^{q(|x|)}$. ■

It follows that

Theorem 6.50 *Assuming the existence of (nonuniformly) one-way function, every NP relation has a zero-knowledge system for proofs of knowledge.*

6.7.3 Applications

We briefly review some of the applications for (zero-knowledge) proofs of knowledge. Typically, (zero-knowledge) proofs of knowledge are used for "mutual disclosure" of the same information. Suppose that `Alice` and `Bob` both claim that they know something (e.g., a 3-coloring of a common input) but are each doubtful of the other person's claim. Employing a zero-knowledge proof of knowledge in both direction is indeed a (conceptually) simple solution to the problem of convincing each other of their knowledge.

Non-oblivious commitment schemes

When using a commitment scheme the receiver is guaranteed that after the commit phase the sender is committed to at most one value (in the sense that it can later "reveal" only this value). Yet, the receiver is not guaranteed that the sender "knows" to what value it is committed. Such a guarantee may be useful in many settings, and can be obtained by using proof of knowledge. For more details see Subsection 6.9.2.

Chosen message attacks

An obvious way of protecting against chosen message attacks on a (public-key) encryption scheme is to augment the ciphertext by a zero-knowledge proof of knowledge of the cleartext. (For definition and alternative constructions of such schemes see Section `[missing(enc-strong.sec)]`.) However, one should note that the resulting encryption scheme employs bidirectional communication between the sender and the receiver (of the encrypted message). It seems that the use of *non-interactive* zero-knowledge proofs of knowledge would yield unidirectional (public-key) encryption schemes. Such claims have been made, yet no proof has ever appeared (and we refrain from expressing an opinion on the issue). Non-interactive zero-knowledge proofs are discussed in Section 6.10.

A zero-knowledge proof system for GNI

The interactive proof of Graph Non-Isomorphism (GNI), presented in Construction 6.8, is not zero-knowledge (unless $GNI \in \mathcal{BPP}$). A cheating verifier may construct a graph H and learn whether it is isomorphic to the first input graph by sending H as query to the prover. A more appealing refutation can be presented to the claim that Construction 6.8 is auxiliary-input zero-knowledge (e.g., the verifier can check whether its auxiliary-input is isomorphic to one of the common-input graphs). We observe however, that Construction 6.8 "would have been zero-knowledge" if the verifier always knew the answer to its queries (as is the case for the honest verifier). The idea then is to have the verifier prove to the prover that he (i.e., the verifier) knows the answer to the query (i.e., an isomorphism to the appropriate input graph), and the prover answers the query only if it is convinced of this claim. Certainly, the verifier's proof of knowledge should not yield the answer (otherwise the prover can use this information in order to cheat thus foiling the soundness requirement). If the verifier's proof of knowledge is zero-knowledge then certainly it does not yield the answer. In fact, it suffices that the verifier's proof of knowledge is *witness-independent* (see Section 6.6).

6.7.4 Proofs of Identity (Identification schemes)

Identification schemes are useful in large distributed systems in which the users are not acquainted with one another. A typical, everyday example is the consumer-retailer situation. In computer systems, a typical example is electronic mail (in communication networks containing sites allowing too loose local super-user access). In between, in technological sophistication, are the Automatic Teller Machine (ATM) system. In these distributed systems, one wishes to allow users to be able to authenticate themselves to other users. This goal is achieved by identification schemes, defined below. In the sequel, we shall also see that identification schemes are intimately related to proofs of knowledge. We just hint that a person's identity can be linked to his ability to do something, and in particular to his ability to prove knowledge of some sort.

Definition

Loosely speaking, an identification scheme consists of a *public file* containing *records* for each user and an *identification protocol*. Each record consists of the name (or identity) of a user and auxiliary *identification information* to be used when invoking the identification protocol (as discussed below). The public file is established and maintained by a trusted party which vouches for the authenticity of the records (i.e., that each record has been submitted by the user the name of which is specified in it). All users have read access to the public file at all times. Alternatively, the trusted party can supply each user with a

signed copy of its public record. Suppose now, that `Alice` wishes to prove to `Bob` that it is indeed her communicating with him. To this end, `Alice` invokes the identification protocol with the (public file) record corresponding to her name as a parameter. `Bob` verifies that the parameter in use indeed matches `Alice`'s public record and proceeds executing his role in the protocol. It is required that `Alice` will always be able to convince `Bob` (that she is indeed `Alice`), whereas nobody else can fool `Bob` into believing that she/he is `Alice`. Furthermore, `Carol` should not be able to impersonate as `Alice` even after receiving polynomially many proofs of identity from `Alice`.

Clearly, if the identification information is to be of any use, then `Alice` must keep in secret the random coins she has used to generate her record. Furthermore, `Alice` must use these stored coins, during the execution of the identification protocol, but this must be done in a way which does not allow her counterparts to later impersonate her.

Conventions: In the following definition we adopt the formalism and notations of interactive machines with auxiliary input (presented in Definition 6.10). We recall that when M is an interactive machine, we denote by $M(y)$ the machine which results by fixing y to be the auxiliary input of machine M. In the following definition n is the security parameter, and we assume with little loss of generality, that the names (i.e., identities) of the users are encoded by strings of length n. If A is a probabilistic algorithm and $x, r \in \{0,1\}^*$, then $A_r(x)$ denotes the output of algorithm A on input x and random coins r.

Remark: In first reading, the reader may ignore algorithm A and the random variable T_n in the security condition. Doing so, however, yields a weaker condition, that is typically unsatisfactory.

Definition 6.51 (identification scheme): *An* identification scheme *consists of a pair, (I, Π), where I is a probabilistic polynomial time algorithm and $\Pi = (P, V)$ is a pair of probabilistic polynomial-time interactive machines satisfying the following conditions*

- Viability: *For every $n \in \mathbf{N}$, every $\alpha \in \{0,1\}^n$, and every $s \in \{0,1\}^{\mathrm{poly}(n)}$*

$$\mathrm{Prob}\left(\langle P(s), V\rangle(\alpha, I_s(\alpha)) = 1\right) = 1$$

- Security: *For every pair of probabilistic polynomial-time interactive machines, A and B, every polynomial $p(\cdot)$, all sufficiently large $n \in \mathbf{N}$, every $\alpha \in \{0,1\}^n$, and every z*

$$\mathrm{Prob}\left(\langle B(z, T_n), V\rangle(\alpha, I_{S_n}(\alpha)) = 1\right) < \frac{1}{p(n)}$$

  where S_n is a random variable uniformly distributed over $\{0,1\}^{\mathrm{poly}(n)}$, and T_n is a random variable describing the output of $A(z)$ after interacting with $P(S_n)$ on common input α, for polynomially many times.

Algorithm I is called the information generating algorithm*, and the pair (P, V) is called the* identification protocol*.*

Hence, to use the identification scheme a user, say `Alice`, the identity of which is encoded by the string α, should first uniformly select a secret string s, compute $i \stackrel{\text{def}}{=} I_s(\alpha)$, ask the trusted party to place the record (α, i) in the public file, and store the string s in a safe place. The viability condition asserts that `Alice` can convince `Bob` of her identity by executing the identification: `Alice` invokes the program P using the stored string s as auxiliary input, and `Bob` uses the program V and makes sure that the common input is the public record containing α (which is in the public file). Ignoring, for a moment, algorithm A and the random variable T_n, the security condition yields that it is infeasible for a party to impersonate `Alice` if all this party has is the public record of `Alice` and some unrelated auxiliary input. However, such a security condition may not suffice in many applications since a user wishing to impersonate `Alice` may ask her first to prover her identity to him/her. The (full) security condition asserts that even if `Alice` has proven her identity to `Carol` many times in the past, still it is infeasible for `Carol` to impersonate `Alice`. We stress that `Carol` cannot impersonate `Alice` to `Bob` provided that she cannot interact concurrently with both. In case this condition does not hold then nothing is guaranteed (and indeed `Carol` can easily cheat by referring `Bob`'s questions to `Alice` and answering as `Alice` does).

Identification schemes and proofs of knowledge

A natural way of establishing a person's identity is to ask him/her to supply a proof of knowledge of a fact that this person is supposed to know. Let us consider a specific (and in fact quite generic) example.

Construction 6.52 (identification scheme based on a one-way function): *Let f be a function. On input an identity $\alpha \in \{0,1\}^n$, the* information generating algorithm *uniformly selects a string $s \in \{0,1\}^n$ and outputs $f(s)$.* (The pair $(\alpha, f(s))$ is the public record for the user with name α). *The* identification protocol *consists of a proof of knowledge of the inverse of the second element in the public record. Namely, in order to prove its identity, user α proves that he knows a string s so that $f(s) = r$, where (α, r) is a record in the public file. (The proof of knowledge in used is allowed to have negligible knowledge error.)*

Proposition 6.53 *If f is a one-way function and the proof of knowledge in use is zero-knowledge then Construction 6.52 constitutes an identification scheme.*

Hence, identification schemes exist if one-way functions exist. More efficient identification schemes can be constructed based on specific intractability assumptions. For example, assuming the intractability of factoring, the so called Fiat-Shamir identification scheme, which is actually a proof of knowledge of a square root, follows.

Construction 6.54 (the Fiat-Shamir identification scheme): *On input an identity* $\alpha \in \{0,1\}^n$, *the* information generating algorithm *uniformly selects a composite number* N, *which is the product of two* n*-bit long primes, a residue* s *mod* N, *and outputs the pair* $(N, s^2 \bmod N)$. (The pair $(\alpha, (N, s^2 \bmod N))$ is the public record for user α). *The* identification protocol *consists of a proof of knowledge of the corresponding modular square root. Namely, in order to prove its identity, user* α *proves that he knows a square root of* $r \stackrel{\text{def}}{=} s^2 \bmod N$, *where* $(\alpha, (r, N))$ *is a record in the public file. (Again, negligible knowledge error is allowed.)*

The proof of knowledge of square root is analogous to the proof system for Graph Isomorphism presented in Construction 6.16. Namely, in order to prove knowledge of a square root of $r \equiv s^2 \pmod N$, the prover repeats the following steps sufficiently many times:

Construction 6.55 (atomic proof of knowledge of square root):

- *The prover randomly selects a residue,* q, *modulo* N *and send* $t \stackrel{\text{def}}{=} q^2 \bmod N$ *to the verifier;*
- *The verifier uniformly selects* $\sigma \in \{0,1\}$ *and sends it to the prover;*
- *Motivation: in case* $\sigma = 0$ *the verifier asks for a square root of* $t \bmod N$, *whereas in case* $\sigma = 1$ *the verifier asks for a square root of* $t \cdot r \bmod N$. *In the sequel we assume, without loss of generality, that* $\sigma \in \{0,1\}$.
- *The prover replies with* $p \stackrel{\text{def}}{=} q \cdot s^\sigma \bmod N$;
- *The verifier accepts (this time) if and only if the messages* t *and* p *sent by the prover satisfies* $p^2 \equiv t \cdot r^\sigma \bmod N$;

When Construction 6.55 is repeated k times, either sequentially or in parallel, the resulting protocol constitutes a proof of knowledge of modular square root with knowledge error 2^{-k}. In case these repetitions are conducted sequentially, then the resulting protocol is zero-knowledge. Yet, for use in Construction 6.54 it suffices that the proof of knowledge is *witness-hiding*, and fortunately even polynomially many parallel executions can be shown to be witness-hiding (see Section 6.6). Hence the resulting identification scheme has constant round complexity. We remark that for identification purposes it suffices to perform Construction 6.55 superlogarithmically many times. Furthermore, also less repetitions are of value: when applying Construction 6.55 $k = O(\log n)$ times, and using the resulting protocol in Construction 6.54, we get a scheme (for identification) in which impersonation can occur with probability at most 2^{-k}.

Identification schemes and proofs of ability

As hinted above, a proof of knowledge of a string (i.e., the ability to output the string) is a special case of a proof of ability to do something. It turns out that identification schemes can be based also on the more general concept of proofs of ability. We avoid defining this concept, and refrain ourself to two "natural" examples of using a proof of ability as basis for identification.

It is an everyday practice to identify people by their ability to produce their signature. This practice can be carried into the digital setting. Specifically, the public record of `Alice` consists of her name and the verification key corresponding to her secret signing key in a predetermined signature scheme. The identification protocol consists of `Alice` signing a random message chosen by the verifier.

A second popular means of identification consists of identifying people by their ability to answer correctly personal questions. A digital analogue to this practice follows. To this end we use pseudorandom functions (see Section 3.6) and zero-knowledge proofs (of membership in a language). The public record of `Alice` consists of her name and a "commitment" to a randomly selected pseudorandom function (e.g., either via a string-commitment to the index of the function or via a pair consisting of a random domain element and the value of the function at this point). The identification protocol consists of `Alice` returning the value of the function at a random location chosen by the verifier, and supplying a zero-knowledge proof that the value returned indeed matches the function appearing in the public record. We remark that the digital implementation offers more security than the everyday practice. In the everyday setting the verifier is given the list of all possible question and answer pairs and is trusted not to try to impersonate as the user. Here we replaced the possession of the correct answers by a zero-knowledge proof that the answer is correct.

6.8 * Computationally-Sound Proofs (Arguments)

In this section we consider a relaxation of the notion of an interactive proof system. Specifically, we relax the soundness condition of interactive proof systems. Instead of requiring that it is *impossible* to fool the verifier into accepting false statement (with probability greater than some bound), we only require that it is *infeasible* to do so. We call such protocols *computationally sound proof systems* (or *arguments*). The advantage of computationally sound proof systems is that *perfect* zero-knowledge *computationally sound* proof systems can be constructed, under some reasonable complexity assumptions, for all languages in $\mathcal{NP}$. We remark that *perfect* zero-knowledge proof systems are unlikely to exists for all languages in $\mathcal{NP}$ (see section 6.5). We recall that *computational* zero-knowledge proof systems do exist for all languages in $\mathcal{NP}$, provided that one-way functions exist. Hence, the above quoted positive results exhibit some kind of a trade-off between the soundness and zero-knowledge

properties of the zero-knowledge protocols of $\mathcal{NP}$. We remark, however, that this is not a real trade-off since the perfect zero-knowledge computationally sound proofs for $\mathcal{NP}$ are constructed under stronger complexity theoretic assumption than the ones used for the computationally zero-knowledge proofs. It is indeed an interesting research project to try to construct perfect zero-knowledge computationally sound proofs for $\mathcal{NP}$ under weaker assumptions (and in particular assuming only the existence of one-way functions).

We remark that it seems that computationally-sound proof systems can be much more efficient than ordinary proof systems. Specifically, under some plausible complexity assumptions, extremely efficient computationally-sound proof systems (i.e., requiring only poly-logarithmic communication and randomness) exist for any language in $\mathcal{NP}$. An analogous result cannot hold for ordinary proof systems, unless $\mathcal{NP}$ is contained in deterministic quasi-polynomial time (i.e., $\mathcal{NP} \subseteq \mathrm{Dtime}(2^{\mathrm{polylog}})$).

6.8.1 Definition

The definition of computationally sound proof systems follows naturally from the above discussion. The only issue to consider is that merely replacing the soundness condition of Definition 6.4 by the following *computational soundness* condition leads to an unnatural definition, since the computational power of the prover in the completeness condition (in Definition 6.4) is not restricted.

> *Computational Soundness:* For every polynomial-time interactive machine B, and for all sufficiently long $x \notin L$
>
> $$\mathrm{Prob}\left(\langle B, V\rangle(x) = 1\right) \leq \frac{1}{3}$$

Hence, it is natural to restrict the prover in both (completeness and soundness) conditions to be an efficient one. It is crucial to interpret efficient as being probabilistic polynomial-time *given auxiliary input* (otherwise only languages in $\mathcal{BPP}$ will have such proof systems). Hence, our starting point is Definition 6.10 (rather than Definition 6.4).

Definition 6.56 (computationally sound proof system) (arguments): *A pair of interactive machines, (P, V), is called an* computationally sound proof system for a language *L if both machines are polynomial-time* (with auxiliary inputs) *and the following two conditions hold*

- Completeness: *For every $x \in L$ there exists a string y such that for every string z*

  $$\mathrm{Prob}\left(\langle P(y), V(z)\rangle(x) = 1\right) \geq \frac{2}{3}$$

- Computational Soundness: *For every polynomial-time interactive machine B, and for all sufficiently long $x \notin L$ and every y and z*

$$\text{Prob}\left(\langle B(y), V(z)\rangle(x) = 1\right) \leq \frac{1}{3}$$

As usual, the error probability in the completeness condition can be reduced (from $\frac{1}{3}$) up to $2^{-\text{poly}(|x|)}$, by repeating the protocol sufficiently many times. The same is *not* true, in general, with respect to the error probability in the computational soundness condition (see Exercise 21). All one can show is that the error probability can be reduced to be negligible (i.e., smaller that $1/p(\cdot)$, for every polynomial $p(\cdot)$). Specifically, by repeating a computationally sound proof sufficiently many time (i.e., superlogarithmically many times) we get a new verifier V' for which it holds that

> For every polynomial $p(\cdot)$, every polynomial-time interactive machine B, and for all sufficiently long $x \notin L$ and every y and z
>
> $$\text{Prob}\left(\langle B(y), V'(z)\rangle(x) = 1\right) \leq \frac{1}{p(|x|)}$$

See Exercise 20.

6.8.2 Perfect Commitment Schemes

The thrust of the current section is in a method for constructing *perfect* zero-knowledge *arguments* for every language in $\mathcal{NP}$. This method makes essential use of the concept of commitment schemes with a *perfect* (or "information theoretic") secrecy property. Hence, we start with an exposition of "perfect" commitment schemes. We remark that such schemes may be useful also in other settings (e.g., in settings in which the receiver of the commitment is computationally unbounded, see for example Section 6.9).

The difference between commitment scheme (as defined in Subsection 6.4.1) and perfect commitment schemes (defined below) consists of a switching in scope of the secrecy and unambiguity requirements. In commitment schemes (see Definition 6.20), the secrecy requirement is computational (i.e., refers only to probabilistic polynomial-time adversaries), whereas the unambiguity requirement is information theoretic (and makes no reference to the computational power of the adversary). On the other hand, in perfect commitment schemes (see definition below), the secrecy requirement is information theoretic, whereas the unambiguity requirement is computational (i.e., refers only to probabilistic polynomial-time adversaries). Hence, in some sense calling one of these schemes "perfect" is somewhat unfair to the other (yet, we do so in order to avoid cumbersome terms as a "perfectly-secret/computationally-nonambiguous commitment scheme"). We remark that it is impossible to have a commitment scheme in which both the secrecy and unambiguity requirements are information theoretic (see Exercise 22).

Definition

Loosely speaking, a perfect commitment scheme is an efficient *two-phase* two-party protocol through which the *sender* can commit itself to a *value* so the following two conflicting requirements are satisfied.

1. *Secrecy*: At the end of the *commit phase* the *receiver* does not gain any *information* of the sender's value.
2. *Unambiguity*: It is infeasible for the sender to interact with the receiver so that the commit phase is successfully terminated and yet later it is feasible for the sender to perform the *reveal phase* in two different ways leading the receiver to accept (as legal "openings") two different values.

Using analogous conventions to the ones used in Subsection 6.4.1, we make the following definition.

Definition 6.57 (perfect bit commitment scheme)*: A* perfect bit commitment scheme *is a pair of probabilistic polynomial-time interactive machines, denoted* (S, R) *(for* sender *and* receiver*), satisfying:*

- Input Specification: *The common input is an integer* n *presented in unary (serving as the security parameter). The private input to the sender is a bit* v*.*
- Secrecy: *For every probabilistic (not necessarily polynomial-time) machine* R^* *interacting with* S*, the random variables describing the output of* R^* *in the two cases, namely* $\langle S(0), R^*\rangle(1^n)$ *and* $\langle S(1), R^*\rangle(1^n)$*, are statistically close.*
- Unambiguity:
  Preliminaries. For simplicity $v \in \{0,1\}$ *and* $n \in \mathbb{N}$ *are implicit in all notations. Fix any probabilistic polynomial-time algorithm* F^**.*
  - *As in Definition 6.20, a* receiver's view of an interaction *with the sender, denoted* $(r, \overline{m})$*, consists of the random coins used by the receiver* (r) *and the sequence of messages received from the sender* $(\overline{m})$*. A* sender's view *of the same interaction, denoted* $(s, \tilde{m})$*, consists of the random coins used by the sender* (s) *and the sequence of messages received from the receiver* $(\tilde{m})$*. A* joint view *of the interaction is a pair consisting of corresponding receiver and sender views of the same interaction.*
  - *Let* $\sigma \in \{0,1\}$*. We say that a joint view (of an interaction),* $t \stackrel{\text{def}}{=} ((r, \overline{m}), (s, \tilde{m}))$*, has* a feasible σ-opening (with respect to F^*) *if on input* (t, σ)*, algorithm* F^* *outputs (say, with probability* $> 1/2$*) a string* s' *such that* $\overline{m}$ *describes the messages*

received by R when R uses local coins r and interacts with machine S which uses local coins s' and input $(\sigma, 1^n)$.
(Remark: We stress that s' may, but need not, equal s. The output of algorithm F^* has to satisfy a relation which depends only on the receiver's view part of the input; the sender's view is supplied to algorithm F^* as additional help.)

- *We say that a joint view is* ambiguous (with respect to F^*) *if it has both a feasible* 0*-opening and a feasible* 1*-opening* (w.r.t. F^*).

The unambiguity requirement *asserts that, for all but a negligible fraction of the coin tosses of the receiver, it is infeasible for the sender to interact with the receiver so that the resulting joint view is ambiguous with respect to some probabilistic polynomial-time algorithm F^*. Namely, for every probabilistic polynomial time interactive machine S^*, probabilistic polynomial-time algorithm F^*, polynomial $p(\cdot)$, and all sufficiently large n, the probability that the joint view of the interaction between R and with S^*, on common input 1^n, is ambiguous with respect to F^*, is at most $1/p(n)$.*

In the formulation of the unambiguity requirement, S^* describes the (cheating) sender strategy in the commit phase, whereas F^* describes its strategy in the reveal phase. Hence, it is justified (and in fact necessary) to pass the sender's view of the interaction (between S^* and R) to algorithm F^*. The unambiguity requirement asserts that any efficient strategy S^* will fail to produce a joint view of interaction, which can be latter (efficiently) opened in two different ways supporting two different values. As usual, events occurring with negligible probability are ignored.

As in Definition 6.20, the secrecy requirement refers explicitly to the situation at the end of the commit phase, whereas the unambiguity requirement implicitly assumes that the reveal phase takes the following form:

1. the sender sends to the receiver its initial private input, v, and the random coins, s, it has used in the commit phase;

2. the receiver verifies that v and s (together with the coins (r) used by R in the commit phase) indeed yield the messages that R has received in the commit phase. Verification is done in polynomial-time (by running the programs S and R).

Construction based on one-way permutations

Perfect commitment schemes can be constructed using any one-way permutation. The known scheme, however, involve a linear (in the security parameter) number of rounds. Hence, it can be used for the purposes of the current section, but not for the construction in Section 6.9.

Construction 6.58 (perfect bit commitment): *Let f be a permutation, and $b(x,y)$ denote the inner-product mod 2 of x and y (i.e., $b(x,y) = \sum_{i=1}^{n} x_i y_i \bmod 2$).*

1. commit phase *(using security parameter n):*
   - *The receiver randomly selects $n-1$ linearly indepndent vectors $r^1, ..., r^{n-1} \in \{0,1\}^n$. The sender uniformly selects $s \in \{0,1\}^n$ and computes $y = f(s)$. (So far no message is exchanged between the parties.)*
   - *The parties proceed in $n-1$ rounds. In the i^{th} round ($i = 1, ..., n-1$), the receiver sends r^i to the sender, which replies by computing and sending $c^i \stackrel{\text{def}}{=} b(y, r^i)$.*
   - *At this point there are exactly two solutions to the equations $b(y, r^i) = c^i$, $1 \leq i \leq n-1$. Define $j = 0$ if y is the lexicographically first solution (among the two), and $j = 1$ otherwise. To commit to a value $v \in \{0,1\}$, the sender sends $c^n \stackrel{\text{def}}{=} j \oplus v$ to the receiver.*
2. reveal phase: *In the reveal phase, the sender reveals the string s used in the commit phase. The receiver accepts the value v if $f(s) = y$, $b(y, r^i) = c^i$ for all $1 \leq i \leq n-1$, and y is the lexicographically first solution to these $n-1$ equations iff $c^n = v$.*

Proposition 6.59 *Suppose that f is a one-way permutation. Then, the protocol presented in Construction 6.58 constitutes a perfect bit commitment scheme.*

It is quite easy to see that Construction 6.58 satisfies the secrecy condition. The proof that the unambiguity requirement is satisfied is quite complex and is omitted for space considerations.

Construction based on clawfree collections

Perfect commitment schemes (of constant number of rounds) can be constructed using a strong intractability assumption; specifically, the existence of clawfree collections (see Subsection 2.4.5). This assumption implies the existence of one-way functions, but it is not known whether the converse is true. Nevertheless, clawfree collections can be constructed under widely believed assumptions such as the intractability of factoring and DLP. Actually, the construction of perfect commitment schemes, presented below, uses a clawfree collection with an additional property; specifically, it is assume that the set of indices of the collection (i.e., the range of algorithm I) can be efficiently recognized (i.e., is in $\mathcal{BPP}$). We remark that such collections do exist under the assumption that DLP is intractable (see Subsection 2.4.5).

Construction 6.60 (perfect bit commitment): *Let (I, D, F) be a triplet of efficient algorithms.*

1. commit phase: *To receive a commitment to a bit (using security parameter n), the receiver randomly generates $i = I(1^n)$ and sends it to the sender. To commit to value $v \in \{0,1\}$ (upon receiving the message i from the receiver), the sender checks if indeed i is in the range of $I(1^n)$, and if so the sender randomly generates $s = D(i)$, computes $c = F(v,i,s)$, and sends c to the receiver. (In case i is not in the range of $I(1^n)$ the sender aborts the protocol announcing that the receiver is cheating.)*

2. reveal phase: *In the reveal phase, the sender reveals the string s used in the commit phase. The receiver accepts the value v if $F(v,i,s) = c$, where (i,c) is the receiver's (partial) view of the commit phase.*

Proposition 6.61 *Let (I,D,F) be a clawfree collection with a probabilistic polynomial-time recognizable set of indices (i.e., range of algorithm I). Then, the protocol presented in Construction 6.60 constitutes a perfect bit commitment scheme.*

Proof: The secrecy requirement follows directly from Property (2) of a clawfree collection (combined with the test $i \in I(1^n)$ conducted by the sender). The unambiguity requirement follows from Property (3) of a clawfree collection, using a standard reducibility argument. ■

We remark that the Factoring Clawfree Collection, presented in Subsection 2.4.5, can be used to construct a perfect commitment scheme although this collection *is not known* to have an efficiently recognizable index set. Hence, perfect commitment schemes exists also under the assumption that factoring Blum integers is intractable. Loosely speaking, this is done by letting the receiver prove to the sender (in zero-knowledge) that the selected index, N, satisfies the secrecy requirement. What is actually being proven is that half of the square roots, of each quadratic residue mod N, have Jacobi symbol 1 (relative to N). A zero-knowledge proof system of this claim does exist (without assuming anything). We remark that the idea just presented can be described as replacing the requirement that the index set is efficiently recognizable by a zero-knowledge proof that a string is indeed a legitimate index.

Commitment Schemes with a posteriori secrecy

We conclude the discussion of perfect commitment schemes by introducing a relaxation of the secrecy requirement. The resulting scheme cannot be used for the purposes of the current section, yet it is useful in different settings. The advantage in the relaxation is that it allows to construct commitment schemes using any clawfree collection, thus waiving the additional requirement that the index set is efficiently recognizable.

Loosely speaking, we relax the secrecy requirement of perfect commitment schemes by requiring that it only holds whenever the receiver follows it prescribed program (denoted

R). This seems strange since we don't really want to assume that the real receiver follows the prescribed program (but rather allow it to behave arbitrarily). The point is that a real receiver may disclose the coin tosses used by it in the commit phase in a later stage, say even after the reveal phase, and by doing so a posteriori prove that (at least in some weak sense) it was following the prescribed program. Actually, the receiver only proves that he behaved in a manner which is consistent with its program.

Definition 6.62 (commitment scheme with perfect a posteriori secrecy)*: A* bit commitment scheme with perfect a posteriori secrecy *is defined as in Definition 6.8.2, except that the secrecy requirement is replaced by the following* a posteriori secrecy requirement*: For every string $r \in \{0,1\}^{\mathrm{poly}(n)}$ it holds that $\langle S(0), R_r\rangle(1^n)$ and $\langle S(1), R_r\rangle(1^n)$ are statistically close, where R_r denotes the execution of the interactive machine R when using internal coin tosses r.*

Proposition 6.63 *Let (I, D, F) be a clawfree collection. Consider a modification of Construction 6.60, in which the sender's check, of whether i is in the range of $I(1^n)$, is omitted (from the commit phase). Then the resulting protocol constitutes a bit commitment scheme with perfect a posteriori secrecy.*

In contrast to Proposition 6.61, here the clawfree collection may not have an efficiently recognizable index set. Hence, the verifier's check must have been omitted. Yet, the receiver can later prove that the message sent by it during the commit phase (i.e., i) is indeed a valid index by disclosing the random coins it has used in order to generate i (using algorithm I).

Proof: The a posteriori secrecy requirement follows directly from Property (2) of a clawfree collection (combined with the assumption that i in indeed a valid index). The unambiguity requirement follows as in Proposition 6.61. ■

A typical application of commitment scheme with perfect a posteriori secrecy is presented in Section 6.9. In that setting the commitment scheme is used inside an interactive proof with the verifier playing the role of the sender (and the prover playing the role of the receiver). If the verifier a posteriori learns that the prover has been cheating then the verifier rejects the input. Hence, no damage is caused, in this case, by the fact that the secrecy of the verifier's commitments might have been breached.

Nonuniform computational unambiguity

Actually, for the applications to proof/argument systems, both the one below and the one in Section 6.9, we need commitment schemes with perfect secrecy and *nonuniform* computational unambiguity. (The reasons for this need are analogous to the case of the

zero-knowledge proof for $\mathcal{NP}$ presented in Section 6.4.) By nonuniform computational unambiguity we mean that the unambiguity condition should hold also for (nonuniform) families of polynomial-size circuits. We stress that all the constructions of perfect commitment schemes possess the nonuniform computational unambiguity, provided that the underlying clawfree collections foil also nonuniform polynomial-size claw-forming circuits.

In order to prevent the terms of becoming too cumbersome we omit the phrase "nonuniform" when referring to the perfect commitment schemes in the description of the two applications.

6.8.3 Perfect Zero-Knowledge Arguments for NP

Having perfect commitment scheme at our disposal, we can construct perfect zero-knowledge arguments for $\mathcal{NP}$, by modifying the construction of (computational) zero-knowledge proofs (for $\mathcal{NP}$) in a totally syntactic manner. We recall that in these proof systems (e.g., Construction 6.25 for Graph 3-Colorability) the prover uses a commitment scheme in order to commit itself to many values, part of them it later reveals upon the verifier's request. All that is needed is to replace the commitment scheme used by the prover by a perfect commitment scheme. We claim that the resulting protocol is a perfect zero-knowledge argument (computationally sound proof) for the original language. For sake of concreteness we prove

Proposition 6.64 *Consider a modification of Construction 6.25 so that the commitment scheme used by the prover is replaced by a perfect commitment scheme. Then the resulting protocol is a perfect zero-knowledge weak argument for Graph 3-Colorability.*

By a weak argument we mean a protocol in which the gap between the completeness and the computational soundness condition is non-negligible. In our case the verifier always accepts inputs in $G3C$, whereas no efficient prover can fool him into accepting graphs $G=(V,E)$ not in $G3C$ with probability greater than $1-\frac{1}{2|E|}$. We remind the reader that by polynomially many repetitions the error probability can be made negligible.

Proof Sketch: We start by proving that the resulting protocol is perfect zero-knowledge for $G3C$. We use the same simulator as in the proof of Proposition 6.26. However, this time analyzing the properties of the simulator is much easier since the commitments are distributed independently of the committed values, and consequently the verifier acts in total oblivion of the values. It follows that the simulator outputs a transcript with probability exactly $\frac{2}{3}$, and for similar reasons this transcript is distributed identically to the real interaction. The perfect zero-knowledge property follows.

The completeness condition is obvious as in the proof of Proposition 6.26. It is left to prove that the protocol satisfies the computational soundness requirement. This is indeed the more subtle part of the current proof (in contrast to the proof of Proposition 6.26 in

which proving soundness is quite easy). We use a reducibility argument to show that a prover's ability to cheat with too high probability on inputs not in $G3C$ translates to an algorithm contradicting the unambiguity of the commitment scheme. Details follows.

We assume, to the contradiction, that there exists a (polynomial-time) cheating prover P^*, and an infinite sequence integers, so that for each integer n there exists graphs $G_n = (V_n, E_n) \notin G3C$ and a string y_n so that $P^*(y_n)$ leads the verifier to accept G_n with probability $> 1 - \frac{1}{2|E_n|}$. Let $k \stackrel{\text{def}}{=} |V_n|$. Let $c_1, ..., c_k$ be the sequence of commitments (to the vertices colors) sent by the prover in step (P1). Recall that in the next step, the verifier sends a uniformly chosen edge (of E_n) and the prover must answer by revealing different colors for its endpoint, otherwise the verifier rejects. A straightforward calculation shows that, since G_n is not 3-colorable, there must exist a vertex for which the prover is able to reveal at least two different colors. Hence, we can construct a polynomial-size circuit, incorporating P^*, G_n and y_n, that violates the (nonuniform) unambiguity condition. Contradiction to the hypothesis of the proposition follows, and this completes the proof. ■

Combining Propositions 6.59 and 6.64, we get

Corollary 6.65 *If non-uniformly one-way permutations exist then every language in $\mathcal{NP}$ has a perfect zero-knowledge argument.*

Concluding Remarks

Propositions 6.26 and 6.64 exhibit a kind of a trade-off between the strength of the soundness and zero-knowledge properties. The protocol of Proposition 6.26 offers computational zero-knowledge and "perfect" soundness, whereas the protocol of Proposition 6.64 offers perfect zero-knowledge and computational soundness. However, one should note that *the two results are not obtained under the same assumptions.* The conclusion of Proposition 6.26 is valid as long as any one-way functions exist, whereas the conclusion of Proposition 6.64 requires a (probably much) stronger assumption. Yet, one may ask which of the two protocols should we prefer, *assuming that they are both valid.* The answer depends on the setting (i.e., application) in which the protocol is to be used. In particular, one should consider the following issues

- The relative importance attributed to soundness and zero-knowledge in the specific application. In case of clear priori to one of the two properties a choice should be made accordingly.
- The computational resources of the various users in the application. One of the users may be known to be in possession of much more substantial computing resources, and it may be reasonable to require that he/she should not be able to cheat even not in an information theoretic sense.

- The soundness requirement refers only to the duration of the execution, whereas in many applications zero-knowledge may be of concern also for a long time afterwards. If this is the case then perfect zero-knowledge arguments do offer a clear advantage (over zero-knowledge proofs).

6.8.4 Zero-Knowledge Arguments of Polylogarithmic Efficiency

A dramatic improvement in the efficiency of zero-knowledge arguments for $\mathcal{NP}$, can be obtained by combining ideas from Chapter `[missing(sign.sec)]` and a result described in Section `[missing(eff-pcp.sec)]`. In particular, assuming the existence of very strong collision-free hashing functions one can construct a computationally-sound (zero-knowledge) proof, for any language in $\mathcal{NP}$, which uses only polylogarithmic amount of communication and randomness. The interesting point in the above statement is the mere existence of such extremely efficient argument, let alone their zero-knowledge property. Hence, we refrain ourselves to describing the ideas involved in constructing such arguments, and do not address the issue of making them zero-knowledge.

By Theorem `[missing(np-pcp.thm)]`, every $\mathcal{NP}$ language, L, can be reduced to $3SAT$ so that non-members of L are mapped into 3CNF formulae for which every truth assignment satisfies at most an $1-\epsilon$ fraction of the clauses, where $\epsilon > 0$ is a universal constant. Let us denote this reduction by f. Now, in order to prove that $x \in L$ it suffices to prove that the formula $f(x)$ is satisfiable. This can be done by supplying a satisfying assignment for $f(x)$. The interesting point is that the verifier need not check that all clauses of $f(x)$ are satisfied by the given assignment. Instead, it may uniformly select only polylogarithmically many clauses and check that the assignment satisfies all of them. If $x \in L$ (and the prover supplies a satisfying assignment to $f(x)$) then the verifier will always accept. Yet, if $x \notin L$ then no assignment satisfies more than a $1-\epsilon$ fraction of the clauses, and consequently a uniformly chosen clause is not satisfied with probability at least ϵ. Hence, checking superlogarithmically many clauses will do.

The above paragraph explains why the randomness complexity is polylogarithmic, but it does not explain why the same holds for the communication complexity. For this end we need an additional idea. The idea is to use a special commitment scheme which allows to commit to a string of length n so that the commitment phase takes polylogarithmic communication and individual bits of this string can be revealed (and verified correct) at polylogarithmic communication cost. For constructing such a commitment scheme we use a *collision-free* hashing function. The function maps strings of some length to strings of half the length so that it is "hard" to find two strings which are mapped by the function to the same image.

Let n denote the length of the input string to which the sender wishes to commit itself, and let k be a parameter (which is later set to be polylogarithmic in n). Denote by H a collision-free hashing function mapping strings of length $2k$ into strings of length k. The

sender partitions its input string into $m \stackrel{\text{def}}{=} \frac{n}{k}$ consequtive blocks, each of length k. Next, the sender constructs a binary tree of depth $\log_2 m$, placing the m blocks in the corresponding leaves of the tree. In each internal node, the sender places the hash value obtained by applying the function H to the contents of the children of this node. The only message sent in the commit phase is the contents of the root (sent by the sender to the receiver). By doing so, *unless the sender can form collisions under* H, the sender has "committed" itself to some n-bit long string. When the receiver wishes to get the value of a specific bit in the string, the sender reveals to the receiver the contents of *both children* of each node along the path from the root to the corresponding leaf. The receiver checks that the values supplied for each node (along the path) match the value obtained by applying H to the values supplied for the two children.

The protocol for arguing that $x \in L$ consists of the prover committing itself to a satisfying assignment for $f(x)$, using the above scheme, and the verifier checking individual clauses by asking the prover to reveal the values assigned to the variables in these clauses. The protocol can be shown to be computationally-sound provided that it is infeasible to find a pair $\alpha, \beta \in \{0,1\}^{2k}$ so that $H(\alpha) = H(\beta)$. Specifically, we need to assume that forming collisions under H is not possible in subexponential time; namely, that for some $\delta > 0$, forming collisions with probability greater than 2^{-k^δ} must take at least 2^{k^δ} time. In such a case, we set $k = (\log n)^{1+\frac{1}{\delta}}$ and get a computationally-sound proof of communication complexity $O(\frac{\log n}{o(1)} \cdot m \cdot k) = \text{polylog}(n)$. (Weaker lower bounds for the collision-forming task may still yield meaningful results by an appropriate setting of the parameter k.) We stress that collisions can always be formed in time 2^{2k} and hence the entire approach fails if the prover is not computationally bounded (and consequently we cannot get (perfectly-sound) proof systems this way). Furthermore, by a simulation argument one may show that, only languages in $\text{Dtime}(2^{\text{polylog}})$ have proof systems with polylogarithmic communication and randomness complexity.

6.9 * Constant Round Zero-Knowledge Proofs

In this section we consider the problem of constructing *constant-round* zero-knowledge proof systems *with negligible error probability* for all languages in $\mathcal{NP}$. To make the rest of the discussion less cumbersome we define a proof system to be *round-efficient* if it is both constant-round and with negligible error probability.

We present two approaches to the construction of round-efficient zero-knowledge proofs for $\mathcal{NP}$.

1. Basing the construction of round-efficient zero-knowledge proof systems on commitment schemes with perfect secrecy (see Subsection 6.8.2).

2. Constructing (round-efficient zero-knowledge) *computationally-sound* proof systems (see Section 6.8) instead of (round-efficient zero-knowledge) proof systems.

The advantage of the second approach is that round-efficient zero-knowledge computationally-sound proof systems for $\mathcal{NP}$ can be constructed using any one-way function, whereas it is not known whether round-efficient zero-knowledge proof systems for $\mathcal{NP}$ can be constructed under the same general assumption. In particular, we only know how to construct perfect commitment schemes by using much stronger assumptions (e.g., the existence of clawfree permutations).

Both approaches have one fundamental idea in common. We start with an abstract exposition of this common idea. Recall that the *basic* zero-knowledge proof for Graph 3-Colorability, presented in Construction 6.25, consists of a constant number of rounds. However, this proof system has a non-negligible error probability (in fact the error probability is very close to 1). In Section 6.4, it was suggested to reduce the error probability to a negligible one by sequentially applying the proof system sufficiently many times. The problem is that this yields a proof system with a non-constant number of rounds. A natural suggestion is to perform the repetitions of the basic proof in parallel, instead of sequentially. The problem with this "solution" is that it is not known whether that the resulting proof system is zero-knowledge.

> Furthermore, it is known that it is not possible to present, as done in the proof of Proposition 6.26, a single simulator which uses every possible verifier as a black box (see Section 6.5). The source of trouble is that, when playing many versions of Construction 6.25 in parallel, a cheating verifier may select the edge to be inspected (i.e., step (V1)) in each version depending on the commitments sent in all versions (i.e., in step (P1)). Such behaviour of the verifier defeats a simulator analogous to the one presented in the proof of Proposition 6.26.

The way to overcome this difficulty is to "switch" the order of steps (P1) and (V1). But switching the order of these steps enables the prover to cheat (by sending commitments in which only the "query" edges are colored correctly). Hence, a more refined approach is required. The verifier starts by committing itself to one edge-query for each version (of Construction 6.25), then the prover commits itself to the coloring in each version, and only then the verifier reveals its queries and the rest of the proof proceeds as before. The commitment scheme used by the verifier should prevent the prover from predicting the sequence of edges committed to by the verifier. This is the point were the two approaches differ.

1. The first approach utilizes for this purpose a commitment scheme with perfect secrecy. The problem with this approach is that such schemes are known to exists only under

stronger assumption than merely the existence of one-way function. Yet, such schemes do exists under assumptions such as the intractability of factoring integers of special form or the intractability of the discrete logarithm problem.

2. The second approach bounds the computational resources of prospective cheating provers. Consequently, it suffices to utilize, "against" these provers (as commitment receivers), commitment schemes with computational security. We remark that this approach utilizes (for the commitments done by the prover) a commitment scheme with an extra property. Yet, such schemes can be constructed using any one-way function.

We remark that both approaches lead to protocols that are zero-knowledge in a liberal sense (i.e., using expected polynomial-time simulators).

6.9.1 Using commitment schemes with perfect secrecy

For sake of clarity, let us start by presenting a detailed description of the constant-round interactive proof (for Graph 3-Colorability (i.e., $G3C$)) sketched above. This interactive proof employs two different commitment schemes. The first scheme is the simple commitment scheme (with "computational" secrecy) presented in Construction 6.21. We denote by $C_s(\sigma)$ the commitment of the sender, using coins s, to the (ternary) value σ. The second commitment scheme is a commitment scheme with perfect secrecy (see Section 6.8.2). For simplicity, we assume that this scheme has a commit phase in which the receiver sends one message to the sender which then replies with a single message (e.g., the schemes presented in Section 6.8.2). Let us denote by $P_{m,s}(\alpha)$ the commitment of the sender to string α, upon receiving message m (from the receiver) and when using coins s.

Construction 6.66 (A round-efficient zero-knowledge proof for G3C):

- Common Input: *A simple (3-colorable) graph $G = (V, E)$. Let $n \stackrel{\text{def}}{=} |V|$, $t \stackrel{\text{def}}{=} n \cdot |E|$ and $V = \{1, ..., n\}$.*
- Auxiliary Input to the Prover: *A 3-coloring of G, denoted ψ.*
- Prover's preliminary step (P0): *The prover invokes the commit phase of the perfect commit scheme, which results in sending to the verifier a message m.*
- Verifier's preliminary step (V0): *The verifier uniformly and independently selects a sequence of t edges, $\overline{E} \stackrel{\text{def}}{=} ((u_1, v_1), ..., (u_t, v_t)) \in E^t$, and sends the prover a random commitment to these edges. Namely, the verifier uniformly selects $\overline{s} \in \{0,1\}^n$ and sends $P_{m,\overline{s}}(\overline{E})$ to the prover;*

- Motivating Remark: *At this point the verifier is committed to a sequence of t edges. This commitment is of perfect secrecy;*
- Prover's step (P1): *The prover uniformly and independently selects t permutations, $\pi_1, ..., \pi_t$, over $\{1,2,3\}$, and sets $\phi_j(v) \stackrel{\text{def}}{=} \pi_j(\psi(v))$, for each $v \in V$ and $1 \leq j \leq t$. The prover uses the computational commitment scheme to commit itself to colors of each of the vertices according to each 3-coloring. Namely, the prover uniformly and independently selects $s_{1,1}, ..., s_{n,t} \in \{0,1\}^n$, computes $c_{i,j} = C_{s_{i,j}}(\phi_j(i))$, for each $i \in V$ and $1 \leq j \leq t$, and sends $c_{1,1}, ..., c_{n,t}$ to the verifier;*
- Verifier's step (V1): *The verifier reveals the sequence $\overline{E} = ((u_1, v_1), ..., (u_t, v_t))$ to the prover. Namely, the verifier send $(\overline{s}, \overline{E})$ to the prover;*
- Motivating Remark: *At this point the entire commitment of the verifier is revealed. The verifier now expects to receive, for each j, the colors assigned by the j^{th} coloring to vertices u_j and v_j (the endpoints of the j^{th} edge in $\overline{E}$);*
- Prover's step (P2): *The prover checks that the message just received from the verifier is indeed a valid revealing of the commitment made by the verifier at step (V0). Otherwise the prover halts immediately. Let us denote the sequence of t edges, just revealed, by $(u_1, v_1), ..., (u_t, v_t)$. The prover uses the reveal phase of the computational commitment scheme in order to reveal, for each j, the j^{th} coloring of vertices u_j and v_j to the verifier. Namely, the prover sends to the verifier the sequence of quadruples*

$$(s_{u_1,1}, \phi_1(u_1), s_{v_1,1}, \phi_1(v_1)), ..., (s_{u_t,t}, \phi_t(u_t), s_{v_t,t}, \phi_t(v_t))$$

- Verifier's step (V2): *The verifier checks whether, for each j, the values in the j^{th} quadruple constitute a correct revealing of the commitments $c_{u_j,j}$ and $c_{v_j,j}$, and whether the corresponding values are different. Namely, upon receiving $(s_1, \sigma_1, s'_1, \tau_1)$ through $(s_t, \sigma_t, s'_t, \tau_t)$, the verifier checks whether for each j, it holds that $c_{u_j,j} = C_{s_j}(\sigma_j)$, $c_{v_j,j} = C_{s'_j}(\tau_j)$, and $\sigma_j \neq \tau_j$ (and both are in $\{1,2,3\}$). If all conditions hold then the verifier accepts. Otherwise it rejects.*

We first assert that Construction 6.66 is indeed an interactive proof for $G3C$. Clearly, the verifier always accepts a common input in $G3C$. Suppose that the common input graph, $G = (V, E)$, is not in $G3C$. Clearly, each of the "committed colorings" sent by the prover in step (P1) contains at least one illegally-colored edge. Using the perfect secrecy of the commitments sent by the verifier in step (V0), we deduce that at step (P1) the prover has "no idea" which edges the verifier asks to see (i.e., as far as the information available to the prover is concerned, each possibility is equally likely). Hence, although the prover sends the "coloring commitment" after receiving the "edge commitment", the probability that all the "committed edges" have legally "committed coloring" is at most

$$\left(1 - \frac{1}{|E|}\right)^t \approx e^{-n} < 2^{-n}$$

We now turn to show that Construction 6.66 is indeed zero-knowledge (in the liberal sense allowing *expected* polynomial-time simulators). For every probabilistic (expected) polynomial-time interactive machine, V^*, we introduce an expected polynomial-time simulator, denoted M^*. The simulator starts by selecting and fixing a random tape, r, for V^*. Given the input graph G and the random tape r, the commitment message of the verifier V^* is determined. Hence, M^* invokes V^*, on input G and random tape r, and gets the corresponding commitment message, denoted CM. The simulator proceeds in two steps.

S1) *Extracting the query edges*: M^* generates a sequence of $n \cdot t$ random commitments to dummy values (e.g., all values equal 1), and feeds it to V^*. In case V^* replies by revealing correctly a sequence of t edges, denoted $(u_1, v_1), ..., (u_t, v_t)$, the simulator records these edges and proceed to the next step. In case the reply of V^* is not a valid revealing of the commitment message CM, the simulator halts outputting the current view of V^* (e.g., G, r and the commitments to dummy values).

S2) *Generating an interaction that satisfies the query edges* (oversimplified exposition): Let $(u_1, v_1), ..., (u_t, v_t)$ denote the sequence of edges recorded in step (S1). M^* generates a sequence of $n \cdot t$ commitments, $c_{1,1}, ..., c_{n,t}$, so that for each $j = 1, ..., t$, it holds that $c_{u_j,j}$ and $c_{v_j,j}$ are random commitments to two different random values in $\{1, 2, 3\}$ and all the other $c_{i,j}$'s are random commitments to dummy values (e.g., all values equal 1). The underlying values are called a *pseudo-colorings*. The simulator feeds this sequence of commitments to V^*. If V^* replies by revealing correctly the (above recorded) sequence of edges, then M^* can complete the simulation of a "real" interaction of V^* (by revealing the colors of the endpoints of these recorded edges). Otherwise, the entire step is repeated (until success occurs).

In the rest of the description we ignore the possibility that, when invoked in steps (S1) and (S2), the verifier reveals two different edge commitments. Loosely speaking, this practice is justified by the fact that during expected polynomial-time computations such event can occur only with negligible probability (since otherwise it contradicts the computational unambiguity of the commitment scheme used by the verifier).

To illustrate the behaviour of the simulator assume that the program V^* always reveals correctly the commitment done in step (V0). In such a case, the simulator will find out the query edges in step (S1), and using them in step (S2) it will simulate the interaction of V^* with the real prover. Using ideas as in Section 6.4 one can show that the simulation is computational indistinguishable from the real interaction. Note that in this case, step (S2) of the simulator is performed only once.

Consider now a more complex case in which, on each possible sequence of internal coin tosses r, program V^* correctly reveals the commitment done in step (V0) only with probability $\frac{1}{3}$. The probability in this statement is taken over all possible commitments generated to the dummy values (in the simulator step (S1)). We first observe that the

probability that V^* correctly reveals the commitment done in step (V0), after receiving a random commitment to a sequence of pseudo-colorings (generated by the simulator in step (S2)), is approximately $\frac{1}{3}$. (Otherwise, we derive a contradiction to the computational secrecy of the commitment scheme used by the prover.) Hence, the simulator reaches step (S2) with probability $\frac{1}{3}$, and each execution of step (S2) is completed successfully with probability $p \approx \frac{1}{3}$. It follows that the expected number of times that step (S2) is invoked when running the simulator is $\frac{1}{3} \cdot \frac{1}{p} \approx 1$.

Let us now consider the general case. Let $q(G,r)$ denote the probability that, on input graph G and random tape r, *after receiving random commitments to dummy values* (generated in step (S1)), program V^* correctly reveals the commitment done in step (V0). Likewise, we denote by $p(G,r)$ the probability that, (on input graph G and random tape r) *after receiving a random commitment to a sequence of pseudo-colorings* (generated by the simulator in step (S2)), program V^* correctly reveals the commitment done in step (V0). As before the difference between $q(G,r)$ and $p(G,r)$ is negligible (in terms of the size of the graph G), otherwise one derives contradiction to the computational secrecy of the prover's commitment scheme. We conclude that the simulator reaches step (S2) with probability $q \stackrel{\text{def}}{=} q(G,r)$, and each execution of step (S2) is completed successfully with probability $p \stackrel{\text{def}}{=} p(G,r)$. It follows that the expected number of times that step (S2) is invoked when running the simulator is $q \cdot \frac{1}{p}$. Here are the bad news: we cannot guarantee that $\frac{q}{p}$ is approximately 1 or even bounded by a polynomial in the input size (e.g., let $p = 2^{-n}$ and $q = 2^{-n/2}$, then the difference between them is negligible and yet $\frac{q}{p}$ is not bounded by $\text{poly}(n)$). This is why the above description of the simulator is oversimplified and a modification is indeed required.

We make the simulator expected polynomial-time by modifying step (S2) as follows. We add an intermediate step (S1.5), to be performed only if the simulator did not halt in step (S1). The purpose of step (S1.5) is to provide a good estimate of $q(G,r)$. The estimate is computed by repeating step (S1) until a fixed (polynomial in $|G|$) number of correct V^*-reveals are encountered (i.e., the estimate will be the ratio of the number of successes divided by the number of trial). By fixing a sufficiently large polynomial, we can guarantee that with overwhelmingly high probability (i.e., $1 - 2^{-\text{poly}(|G|)}$) the estimate is within a constant factor of $q(G,r)$. It is easily verified that the estimate can be computed within expected time $\text{poly}(|G|)/q(G,r)$. Step (S2) of the simulator is modified by adding a bound on the number of times it is performed, and if none of these executions yield a correct V^*-reveal then the simulator outputs a *special empty interaction*. Specifically, step (S2) will be performed at most $\text{poly}(|G|)/\overline{q}$, where $\overline{q}$ is the estimate to $q(G,r)$ computed in step (S1.5). It follows that the modified simulator has expected running time bounded by $q(G,r) \cdot \frac{\text{poly}(|G|)}{q(G,r)} = \text{poly}(|G|)$.

It is left to analyze the output distribution of the modified simulator. We refrain ourselves to reducing this analysis to the analysis of the output of the original simulator, by bounding the probability that the modified simulator outputs a special empty interaction.

This probability is bounded by

$$\begin{aligned}\Delta(G,r) &\stackrel{\text{def}}{=} q(G,r) - q(G,r)\cdot\left(1-(1-p(G,r))^{\mathrm{poly}(|G|)/q(G,r)}\right)\\ &= q(G,r)\cdot(1-p(G,r))^{\mathrm{poly}(|G|)/q(G,r)}\end{aligned}$$

We claim that $\Delta(G,r)$ is a negligible function of $|G|$. Assume, to the contrary, that there exists a polynomial $P(\cdot)$, an infinite sequence of graphs $\{G_n\}$, and an infinite sequence of random tapes $\{r_n\}$, such that $\Delta(G_n,r_n) > 1/P(n)$. It follows that for each such n we have $q(G_n,r_n) > 1/P(n)$. We consider two cases.

Case 1: For infinitely many n's, it holds that $p(G_n,r_n) \geq q(G_n,r_n)/2$. In such a case we get for these n's

$$\begin{aligned}\Delta(G_n,r_n) &\leq (1-p(G_n,r_n))^{\mathrm{poly}(|G_n|)/q(G_n,r_n)}\\ &\leq \left(1-\frac{q(G_n,r_n)}{2}\right)^{\mathrm{poly}(|G_n|)/q(G_n,r_n)}\\ &< 2^{-\mathrm{poly}(|G_n|)/2}\end{aligned}$$

which contradicts our hypothesis that $\Delta(G_n,r_n) > 1/\mathrm{poly}(n)$.

Case 2: For infinitely many n's, it holds that $p(G_n,r_n) < q(G_n,r_n)/2$. It follows that for these n's we have $|q(G_n,r_n) - p(G_n,r_n)| > P(n)/2$, which leads to contradiction of the computational secrecy of the commitment scheme (used by the prover).

Hence, contradiction follows in both cases.

We remark that one can modify Construction 6.66 so that weaker forms of perfect commitment schemes can be used. We refer specifically to commitment schemes with perfect a posteriori secrecy (see Subsection 6.8.2). In such schemes the secrecy is only established a posteriori by the receiver which discloses the coin tosses it has used in the commit phase. In our case, the prover plays the role of the receiver, and the verifier plays the role of the sender. It suffices to establish the secrecy property a posteriori, since in case secrecy is not establish the verifier may reject. In such a case no harm has been caused since the secrecy of the perfect commitment scheme is used only to establish the soundness of the interactive proof.

6.9.2 Bounding the power of cheating provers

Construction 6.66 can be modified to yield a zero-knowledge computationally sound proof, under the (more general) assumption that one-way functions exist. In the modified protocol, we let the verifier use a commitment scheme with computational secrecy, instead of

the commitment scheme with perfect secrecy used in Construction 6.66. (Hence, both users commit to their messages using commitment scheme with computational secrecy.) Furthermore, the commitment scheme used by the prover must have the extra property that it is infeasible to construct a commitment without "knowing" to what value it commits. Such a commitment scheme is called *non-oblivious*. We start by defining and constructing non-oblivious commitment schemes.

Non-oblivious commitment schemes

The non-obliviousness of a commitment scheme is intimately related to the definition of proof of knowledge (see Section 6.7).

Definition 6.67 (non-oblivious commitment schemes)*: Let (S, R) be a commitment scheme as in Definition 6.20. We say that the commitment scheme is* non-oblivious *if the prescribed receiver, R, constitutes a knowledge-verifier, that is always convinced by S, for the relation*

$$\{((1^n, r, \overline{m}), (\sigma, s)) : \overline{m} = \mathrm{view}_{R(1^n,r)}^{S(\sigma,1^n,s)}\}$$

where, as in Definition 6.20, $\mathrm{view}_{R(1^n,r)}^{S(\sigma,1^n,s)}$ *denotes the messages received by the interactive machine R on input 1^n and local-coins r, when interactive with machine S (that has input $(\sigma, 1^n)$ and uses coins s).*

It follows that the receiver prescribed program, R, may accept or rejects at the end of the commit phase, and that this decision is supposed to reflect the sender's ability to later come up with a legal opening of the commitment (i.e., successfully complete the reveal phase). We stress that non-obliviousness relates mainly to cheating senders, since the prescribed sender has no difficulty to later successfully complete the reveal phase (and in fact during the commit phase S always convinces the receiver of this ability). Hence, *any* sender program (not merely the prescribed S) can be modified so that at the end of the commit phase it (locally) outputs information enabling the reveal phase (i.e., σ and s). The modified sender runs in expected time that is inversely proportional to the probability that the commit phase is completed successfully.

We remark that in an ordinary commitment scheme, at the end of the commit phase, the receiver does not necessarily "know" whether the sender can later successfully conduct the reveal phase. For example, a cheating sender in Construction 6.21 can (undetectedly) perform the commit phase without ability to later successfully perform the reveal phase (e.g., the sender may just send a uniformly chosen string). It is only guaranteed that if the sender follows the prescribed program then the sender can always succeed in the reveal phase. Furthermore, with respect to the scheme presented in Construction 6.23, a cheating sender can (undetectedly) perform the commit phase in a way that it generates a receiver

view which does not have any corresponding legal opening (and hence the reveal phase is doomed to fail). See Exercise 13.

Nevertheless

Theorem 6.68 *If one-way functions exist then there exist non-oblivious commitment schemes with constant number of communication rounds.*

We recall that (ordinary) commitment schemes can be constructed assuming the existence of one-way functions (see Proposition 6.24 and Theorem 3.29). Consider the relation corresponding to such a scheme. Using zero-knowledge proofs of knowledge (see Section 6.7) for the above relation, we get a non-oblivious commitment scheme. (We remark that such proofs do exist under the same assumptions.) However, the resulting commitment scheme has unbounded number of rounds (due to the round complexity of the zero-knowledge proof). We seem to have reached a vicious circle, yet there is a way out. We can use constant-round witness indistinguishable proofs (see Section 6.6), instead of the zero-knowledge proofs. The resulting commitment scheme has the additional property that when applied (polynomially) many times in parallel the secrecy property holds simultaneously in all copies. This fact follows from the Parallel Composition Lemma for witness indistinguishable proofs (see Section 6.6). The simultaneous secrecy of many copies is crucial to the following application.

Modifying Construction 6.66

We recall that we are referring to a modification of Construction 6.66 in which the verifier uses a commitment scheme (with computational secrecy), instead of the commitment scheme with perfect secrecy used in Construction 6.66. In addition, the commitment scheme used by the prover is non-oblivious.

We conclude this section by remarking on how to adopt the argument of the first approach (i.e., of Subsection 6.9.1) to suit our current needs. We start with the claim that the modified protocol is a computationally-sound proof for $G3C$. Verifying that the modified protocol satisfies the completeness condition is easy as usual. We remark that the modified protocol does not satisfy the (usual) soundness condition (e.g., a "prover" of exponential computing power can break the verifier's commitment and generate pseudo-colorings that will later fool the verifier into accepting). Nevertheless, we can show that the modified protocol does satisfy the computational soundness (of Definition 6.56). Namely, we show that for every polynomial $p(\cdot)$, every polynomial-time interactive machine B, and for all sufficiently large graph $G \notin G3C$ and every y and z

$$\mathrm{Prob}\left(\langle B(y), V_{G3C}(z)\rangle(x) = 1\right) \leq \frac{1}{p(|x|)}$$

where V_{G3C} is the verifier program in the modified protocol.

Using the information theoretic unambiguity of the commitment scheme employed by the prover, we can talk of a unique color assignment which is induced by the prover's commitments. Using the fact that this commitment scheme is non-oblivious, it follows that the prover can be modified so that, in step (P1), it outputs the values to which it commits itself at this step. We can now use the computational secrecy of the verifier's commitment scheme to show that the color assignment generated by the prover is almost independent of the verifier's commitment. Hence, the probability that the prover can fool the verifier to accept an input not in the language is non-negligibly greater than what it would have been if the verifier asked random queries after the prover makes its (color) commitments. The computational soundness of the (modified) protocol follows. We remark that we do not know whether the protocol is computationally sound in case the prover uses a commitment scheme that is not guaranteed to be non-oblivious.

Showing that the (modified) protocol is zero-knowledge is even easier than it was in the first approach (i.e., in Subsection 6.9.1). The reason being that when demonstrating zero-knowledge of such protocols we use the secrecy of the prover's commitment scheme and the unambiguity of the verifier's commitment scheme. Hence, only these properties of the commitment schemes are relevant to the zero-knowledge property of the protocols. Yet, the current (modified) protocol uses commitment schemes with relevant properties which are not weaker than the ones of the corresponding commitment schemes used in Construction 6.66. Specifically, the prover's commitment scheme in the modified protocol possess computationally secrecy just like the prover's commitment scheme in Construction 6.66. We stress that this commitment, like the simpler commitment used for the prover in Construction 6.66, has the simultaneous secrecy (of many copies) property. Furthermore, the verifier's commitment scheme in the modified protocol possess "information theoretic" unambiguity, whereas the verifier's commitment scheme in Construction 6.66 is merely computationally unambiguous.

6.10 * Non-Interactive Zero-Knowledge Proofs

Author's Note: *Indeed, this section is missing*

6.10.1 Definition

6.10.2 Construction

6.11 * Multi-Prover Zero-Knowledge Proofs

In this section we consider an extension of the notion of an interactive proof system. Specifically, we consider the interaction of a verifier with several (say, two) provers. The provers

may share an a-priori selected strategy, but it is assumed that they cannot interact with each other during the time period in which they interact with the verifier. Intuitively, the provers can coordinate their strategies prior to, but not during, their interrogation by the verifier.

The notion of multi-prover interactive proof plays a fundamental role in complexity theory. This aspect is not addressed here (but rather postponed to Section `[missing(eff-pcp.sec)]`). In the current section we merely address the zero-knowledge aspects of multi-party interactive proofs. Most importantly, the multi-prover model enables the construction of (perfect) zero-knowledge proof systems for $\mathcal{NP}$, *independent of any* complexity theoretic (or other) *assumptions*. Furthermore, these proof systems can be extremely efficient. Specifically, the on-line computations of all parties can be performed in polylogarithmic time (on a RAM).

6.11.1 Definitions

For sake of simplicity we consider the two-prover model. We remark that more provers do not offer any essential advantages (and specifically, none that interest us in this section). Loosely speaking, a two-prover interactive proof system is a three party protocol, where two parties are provers and the additional party is a verifier. The only interaction allowed in this model is between the verifier and each of the provers. In particular, a prover does not "know" the contents of the messages sent by the verifier to the other prover. The provers do however share a random input tape, which is (as in the one-prover case) "beyond the reach" of the verifier. The two-prover setting is a special case of the *two-partner model* described below.

The two-partner model

The two-party model consists of two *partners* interacting with a third party, called *solitary*. The two partners can agree on their strategies beforehand, and in particular agree on a common uniformly chosen string. Yet, once the interaction with the solitary begins, the partners can no longer exchange information. The following definition of such an interaction extends Definitions 6.1 and 6.2.

Definition 6.69 (two-partner model): *The* two-partner model *consists of three interactive machines, two are called* partners *and the third is called* solitary*, which are linked and interact as hereby specified.*

- *The input-tapes of all three parties coincide, and its contents is called the* common input*.*
- *The random-tapes of the two partners coincide, and is called the* partners' random-tape*. (The solitary has a separate random-tape.)*

- *The solitary has two pairs of communication-tapes and two switch-tapes; instead of a single pair of communication-tapes and a single switch-tape (as in Definition 6.1).*
- *Both partners have the same identity and the solitary has an opposite identity (see Definitions 6.1 and 6.2).*
- *The first (resp., second) switch-tape of the solitary coincides with the switch-tape of the first (resp., second) partner, the first (resp., second) read-only communication-tape of the solitary coincides with the write-only communication-tape of the first (resp., second) partner and vice versa.*
- *The* joint computation *of the three parties, on a common input x, is a sequence of triplets. Each triplet consists of the local configuration of each of the three machines. The behaviour of each partner-solitary pair is as in the definition of the joint computation of a pair of interactive machines.*
- **Notation:** *We denote by $\langle P_1, P_2, S\rangle(x)$ the output of the solitary S after interacting with the partners P_1 and P_2, on common input x.*

Two-prover interactive proofs

A two-prover interactive proof system is now defined analogously to the one-prover case (see Definitions 6.4 and 6.6).

Definition 6.70 (two-prover interactive proof system): *A triplet of interactive machines, (P_1, P_2, V), in the two-partner model is called an* proof system for a language *L if the machine V* (called verifier) *is probabilistic polynomial-time and the following two conditions hold*

- Completeness: *For every $x \in L$*
$$\mathrm{Prob}\left(\langle P_1, P_2, V\rangle(x) = 1\right) \geq \frac{2}{3}$$
- Soundness: *For every $x \notin L$ and every pair of partners (B_1, B_2),*
$$\mathrm{Prob}\left(\langle B_1, B_2, V\rangle(x) = 1\right) \leq \frac{1}{3}$$

As usual, the error probability in the completeness condition can be reduced (from $\frac{1}{3}$) up to $2^{-\mathrm{poly}(|x|)}$, by *sequentially* repeating the protocol sufficiently many times. We stress that error reduction via *parallel* repetitions is **not** known to work *in general*.

The notion of zero-knowledge (for multi-prove systems) remains exactly as in the one-prover case. Actually, the definition of perfect zero-knowledge may even be made more strict by requiring that the simulator never fails (i.e., never outputs the special symbol $\perp$). Namely,

Definition 6.71 *We say that a* (two-prover) *proof system* (P_1, P_2, V) *for a language* L *is* perfect zero-knowledge *if for every probabilistic polynomial-time interactive machine* V^* *there exists a probabilistic polynomial-time algorithm* M^* *such that for every* $x \in L$ *the random variables* $\langle P_1, P_2, V^* \rangle(x)$ *and* $M^*(x)$ *are identically distributed.*

Extension to the auxiliary-input (zero-knowledge) model is straightforward.

6.11.2 Two-Senders Commitment Schemes

The thrust of the current section is in a method for constructing perfect zero-knowledge two-prover proof systems for every language in $\mathcal{NP}$. This method makes essential use of a commitment scheme *for two senders and one receiver* that posses "information theoretic" secrecy and unambiguity properties. We stress that it is impossible to simultaneously achieve "information theoretic" secrecy and unambiguity properties in the single sender model.

A Definition

Loosely speaking, a two-sender commitment scheme is an efficient *two-phase* protocol for the two-partner model, through which the partners, called *senders*, can commit themselves to a *value* so that the following two conflicting requirements are satisfied.

1. *Secrecy*: At the end of the *commit phase* the solitary, called *receiver*, does not gain any *information* of the senders' value.

2. *Unambiguity*: Suppose that the commit phase is successfully terminated. Then if later the senders can perform the *reveal phase* so that the receiver accepts the value 0 with probability p then they cannot perform the *reveal phase* so that the receiver accepts the value 1 with probability substantially bigger than $1 - p$. (Due to the secrecy requirement and the fact that the senders are computationally unbounded, for every p, the senders can always conduct the commit phase so that they can later reveal the value 0 with probability p and the value 1 with probability $1 - p$.)

Instead of presenting a general definition, we restrict our attention to the special case of two-sender commitment schemes in which only the first sender (and the receiver) takes part in the commit phase, whereas only the second sender takes part in the reveal phase. Furthermore, we assume, without loss of generality, that in the reveal phase the (second) sender sends the contents of the joint random-tape (used by the first sender in the commit phase) to the receiver.

Definition 6.72 (two-sender bit commitment)*: A* two-sender bit commitment scheme *is a triplet of probabilistic polynomial-time interactive machines, denoted* (S_1, S_2, R)*, for the two-partner model satisfying:*

- Input Specification: *The common input is an integer* n *presented in unary, called the* security parameter*. The two partners, called the* senders*, have an auxiliary private input* $v \in \{0,1\}$*.*
- Secrecy: *The 0-commitment and the 1-commitment are identically distributed. Namely, for every probabilistic (not necessarily polynomial-time) machine* R^* *interacting with the first sender (i.e.,* S_1*), the random variables* $\langle S_1(0), R^*\rangle(1^n)$ *and* $\langle S_1(1), R^*\rangle(1^n)$ *are identically distributed.*
- Unambiguity: *Preliminaries. For simplicity* $v \in \{0,1\}$ *and* $n \in \mathbf{N}$ *are implicit in all notations.*
  - *As in Definition 6.20, a* receiver's view of an interaction *with the (first) sender, denoted* $(r, \overline{m})$*, consists of the random coins used by the receiver, denoted* r*, and the sequence of messages received from the (first) sender, denoted* $\overline{m}$*.*
  - *Let* $\sigma \in \{0,1\}$*. We say that the string* s *is a* possible σ-opening *of the receiver's view* $(r, \overline{m})$ *if* $\overline{m}$ *describes the messages received by* R *when* R *uses local coins* r *and interacts with machine* S_1 *which uses local coins* s *and input* $(\sigma, 1^n)$*.*
  - *Let* S_1^* *be an arbitrary program for the first sender. Let* p *be a real, and* $\sigma \in \{0,1\}$*. We say that* p *is an* upper bound on the probability of a σ-opening of the receiver's view of the interaction with S_1^* *if for every random variable* X*, which is statistically independent of the receiver's coin tosses, the probability that* X *is a possible* σ*-opening of the receiver's view of an interaction with* S_1^* *is at most* p*.*
    (The probability is taken over the coin tosses of the receiver, the strategy S_1^* and the random variable X.)
  - *Let* S_1^* *be as above, and, for each* $\sigma \in \{0,1\}$*, let* p_σ *be an upper bound on the probability of a* σ*-opening of the interaction with* S_1^**. We say that* the receiver's view of the interaction with S_1^* is unambiguous *if* $p_0 + p_1 \leq 1 + 2^{-n}$*.*

  The unambiguity requirement *asserts that, for every program for the first sender,* S_1^**, the receiver's interaction with* S_1^* *is unambiguous.*

In the formulation of the unambiguity requirement, the random variables X represent possible strategies of the second sender. These strategies may depend on the random input that is shared by the two senders, but is independent of the receiver's random coins (since information on these coins, if at all, is only sent to the first sender). Actually, the highest possible value of $p_0 + p_1$ is attainable by deterministic strategies for both senders. Thus, it suffices to consider an arbitrary deterministic strategy S_1^* for the first sender and a fixed

σ-opening, denoted s^σ, for the second sender (for each $\sigma \in \{0,1\}$). In this case, the probability is taken only over the receiver coin tosses and we can strengthen the unambiguity condition as follows:

> (strong unambiguity condition) *for every deterministic strategy S_1^*, and every pair of strings (s^0, s^1), the probability that* **for both** $\sigma = 0,1$ *the string s^σ is a σ-opening of the receiver's view of the interaction with S_1^* is bounded above by 2^{-n}.*

In general, in case the sender employ randomized strategies, they determine for each possible coin-tossing of the receiver a pair of probabilities corresponding to their success in a 0-opening and a 1-opening. The unambiguity condition asserts that the average of these pairs, taken over all possible receiver's coin tosses is a pair which sums-up to at most $1 + 2^{-n}$. Intuitively, this means that the senders cannot do more harm than deciding at random (possibly based also on the receiver's message to the first sender) whether to commit to 0 or to 1. Both secrecy and unambiguity requirements are information theoretic (in the sense that no computational restrictions are placed on the adversarial strategies). We stress that we have implicitly assumed that the reveal phase takes the following form:

1. the second sender sends to the receiver the initial private input, v, and the random coins, s, used by the first sender in the commit phase;
2. the receiver verifies that v and s (together with the private coins (r) used by R in the commit phase) indeed yield the messages that R has received in the commit phase. Verification is done in polynomial-time (by running the programs S_1 and R).

A Construction

By the above conventions, it suffices to explicitly describe the commit phase (in which only the first sender takes part).

Construction 6.73 (two-sender bit commitment):

- Preliminaries: *Let π_0, π_1 denote two permutations over $\{0,1,2\}$ so that π_0 is the identity permutation and π_1 is a permutation consisting of a single transposition, say $(1,2)$. Namely, $\pi_1(1)=2$, $\pi_1(2)=1$ and $\pi_1(0)=0$.*
- Common input: *the security parameter n (in unary).*
- A convention: *Suppose that the contents of the senders' random-tape encodes a uniformly selected $\overline{s} = s_1 \cdots s_n \in \{0,1,2\}^n$.*

- Commit Phase:

  1. *The receiver uniformly selects* $\overline{r} = r_1 \cdots r_n \in \{0,1\}^n$ *and sends* $\overline{r}$ *to the first sender.*
  2. *To commit to a bit* σ*, the first sender computes* $c_i \stackrel{\text{def}}{=} \pi_{r_i}(s_i) + \sigma \bmod 3$*, for each* i*, and sends* $c_1 \cdots c_n$ *to the receiver.*

We remark that the *second* sender could have opened the commitment either way if he had known $\overline{r}$ (sent by the receiver to the *first* sender). The point is that the second sender does not "know" $\overline{r}$, and this fact drastically limits its ability to cheat.

Proposition 6.74 *Construction 6.73 constitutes a two-sender bit commitment scheme.*

Proof: The security property follows by observing that for every choice of $\overline{r} \in \{0,1\}^n$, the message sent by the first sender is uniformly distributed over $\{0,1,2\}^n$.

The unambiguity property is proven by contradiction. As a motivation, we first consider the execution of the above protocol when n equals 1 and show that it is impossible for the two senders to be *always* able to open the commitments both ways. Consider two messages, $(0,s^0)$ and $(1,s^1)$, sent by the second sender in the reveal phase so that s^0 is a possible 0-opening and s^1 is a possible 1-opening, both with respect to the receiver's view. We stress that these messages are sent obliviously of the random coins of the receiver, and hence must match all possible receiver's views (or else the opening does not always succeed). It follows that for each $r \in \{0,1\}$, both $\pi_r(s^0)$ and $\pi_r(s^1) + 1 \bmod 3$ must fit the message received by the receiver (in the commit phase) in response to message r sent by it. Hence, $\pi_r(s^0) \equiv \pi_r(s^1) + 1 \pmod 3$ holds, for each $r \in \{0,1\}$. Contradiction follows since no two $s^0, s^1 \in \{0,1,2\}$ can satisfy both $\pi_0(s^0) \equiv \pi_0(s^1) + 1 \pmod 3$ and $\pi_1(s^0) \equiv \pi_1(s^1) + 1 \pmod 3$. (The reason being that the first equality implies $s^0 \equiv s^1 + 1 \pmod 3$ which combined with the second equality yields $\pi_1(s_1 + 1 \bmod 3) \equiv \pi_1(s_1) + 1 \pmod 3$, whereas for every $s \in \{0,1,2\}$ it holds that $\pi_1(s + 1 \bmod 3) \not\equiv \pi_1(s) + 1 \pmod 3$.)

We now turn to the actual proof of the unambiguity property. We first observe that if there exists a program S_1^* so that the receiver's interaction with S_1^* is ambiguous, then there exists also such a deterministic program. Actually, the program is merely a function, denoted f, mapping n-bit long strings into sequences in $\{0,1,2\}^n$. Likewise, the (0-opening and 1-opening) strategies for the second sender can be assumed, without loss of generality, to be deterministic. Consequently, both strategies consist of constant sequences, denoted $\overline{s}^0$ and $\overline{s}^1$, and both can be assumed (with no loss of generality) to be in $\{0,1,2\}^n$.

For each $\sigma \in \{0,1\}$, let p_σ denote the probability that the sequence $\overline{s}^\sigma$ is a possible σ-opening of the receiver's view $(U_n, f(U_n))$, where U_n denotes a random variable uniformly distributed over $\{0,1\}^n$. The contradiction hypothesis implies that $p_0 + p_1 > 1 + 2^{-n}$. Put

in other words, $|R^0| + |R^1| \geq 2^n + 2$, where R^σ denotes the set of all strings $\overline{r} \in \{0,1\}^n$ for which the sequence $\overline{s}^\sigma$ is a possible σ-opening of the receiver's view $(\overline{r}, f(\overline{r}))$. Namely,

$$R^\sigma = \{\overline{r} : (\forall i)\ f_i(\overline{r}) \equiv \pi_{r_i}(s_i^\sigma) + \sigma \pmod 3\}$$

where $\overline{r} = r_1 \cdots r_n$, $\overline{s}^\sigma = s_1^\sigma \cdots s_n^\sigma$, and $f(\overline{r}) = f_1(\overline{r}) \cdots f_n(\overline{r})$. We are going to refute the contradiction hypothesis by showing that the intersection of the sets R^0 and R^1 cannot contain more than a single element.

Claim 6.74.1: Let R^0 and R^1 as defined above. Then $|R^0 \cap R^1| \leq 1$.

proof: Suppose, on the contrary, that $\overline{\alpha}, \overline{\beta} \in R^0 \cap R^1$ (and $\overline{\alpha} \neq \overline{\beta}$). Then, there exist an i such that $\alpha_i \neq \beta_i$, and without loss of generality $\alpha_i = 0$ (and $\beta_i = 1$). By the definition of R^σ it follows that

$$\begin{aligned}
f_i(\overline{\alpha}) &\equiv \pi_0(s_i^0) \pmod 3 \\
f_i(\overline{\beta}) &\equiv \pi_1(s_i^0) \pmod 3 \\
f_i(\overline{\alpha}) &\equiv \pi_0(s_i^1) + 1 \pmod 3 \\
f_i(\overline{\beta}) &\equiv \pi_1(s_i^1) + 1 \pmod 3
\end{aligned}$$

Contradiction follows as in the motivating discussion. □

This completes the proof of the proposition. ■

We remark that Claim 6.74.1 actually yields the strong unambiguity condition (presented in the discussion following Definition 6.72). More importantly, we remark that the proof extends easily to the case in which many instances of the protocol are executed in parallel; namely, the parallel protocol constitutes a two-sender multi-value (i.e., string) commitment scheme.

> **Author's Note:** *The last remark should be elaborated significantly. In addition, it should be stressed that the claim holds also when the second sender is asked to reveal only some of the commitments, as long as this request is indepdendent of the coin tosses used by the receiver during the commit phase.*

6.11.3 Perfect Zero-Knowledge for NP

Two-prover perfect zero-knowledge proof systems for any language in $\mathcal{NP}$ follow easily by modifying Construction 6.25. The modification consists of replacing the bit commitment scheme, used in Construction 6.25, by the two-sender bit commitment scheme of Construction 6.73. Specifically, the modified proof system for Graph Coloring proceeds as follows.

Two-prover atomic proof of Graph Coloring

- The first prover uses the prover's random tape to determine a permutation of the coloring. In order to commit to each of the resulting colors, the first prover invokes (the commit phase of) a two-sender bit commitment, setting the security parameter to be the number of vertices in the graph. (The first prover plays the role of the first sender whereas the verifier plays the role of the receiver.)
- The verifier uniformly selects an edge and sends it to the second prover.
- The second prover reveals the colors of the endpoints of the required edge, by sending the portions of the prover's random-tape used in the corresponding instance of the commit phase.

We now remark on the properties of the above protocol. As usual, one can see that the provers can always convince the verifier of valid claims (i.e., the completeness condition hold). Using the unambiguity property of the two-sender commitment scheme we can think of the first prover as selecting at random, with arbitrary probability distribution, a color assignment to the vertices of the graph. We stress that this claim holds although many instances of the commit protocol are performed concurrently (see remark above). If the graph is not 3-colored than each of the possible color assignments chosen by the first prover is illegal, and a weak soundness property follows. Yet, by executing the above protocol polynomially many times, even in parallel, we derive a protocol satisfying the soundness requirement. We stress that the fact that parallelism is effective here (as means for decreasing error probability) follows from the unambiguity property of two-sender commitment scheme and not from a general "parallel composition lemma" (which is not valid in the two-prover setting).

> **Author's Note:** *The last sentence refers to a false claim by which the error probability of a protocol in which a basic protocol is repeated t times in parallel is at most p^t, where p is the error probability of the basic protocol. Interestingly, Ran Raz has recently proven a general "parallel composition lemma" of slightly weaker form: the error probability indeed decreases exponentially in t (but the base is indeed bigger than p).*

We now turn to the zero-knowledge aspects of the above protocol. It turns out that this part is much easier to handle than in all previous cases we have seen. In the construction of the simulator we take advantage on the fact that it is playing the role of both provers and hence the unambiguity of the commitment scheme does not apply. Specifically, the simulator, playing the role of both senders, can easily open each commitment any way it wants. (Here we take advantage on the specific structure of the commitment scheme of Construction 6.73.) Details follow.

Simulation of the atomic proof of Graph Coloring

- The simulator generates random "commitments to nothing". Namely, the simulator invokes the verifier and answers its messages by uniformly chosen strings.
- Upon receiving the query-egde (u, v) from the verifier, the simulator uniformly selects two different colours, ϕ_u and ϕ_v, and opens the corresponding commitments so that to reveal this values. The simulator has no difficulty to do so since, unlike the second prover, it knows the messages sent by the verifier in the commit phase. (Given the receiver's view, $(r_1 \cdots r_n, c_1 \cdots c_n)$, of the commit phase, a 0-opening is computed by setting $s_i = \pi_{r_i}^{-1}(c_i)$ whereas a 1-opening is computed by setting $s_i = \pi_{r_i}^{-1}(c_i - 1)$, for all i.)

We now remark that the entire argument extends trivially to the case in which polynomially many instances of the protocol are performed concurrently.

Efficiency improvement

A dramatic improvement in the efficiency of two-prover (perfect) zero-knowledge proofs for $\mathcal{NP}$, can be obtained by using the techniques described in Section `[missing(eff-pcp.sec)]`. In particular, such a proof system with constant error probability, can be implemented in probabilistic polynomial-time, so that the number of bits exchanged in the interaction is logarithmic. Furthermore, the verifier is only required to use logarithmically many coin tosses. The error can be reduced to 2^{-k} by repeating the protocol sequentially for k times. In particular negligible error probability is achieved in polylogarithmic communication complexity. We stress again that error reduction via parallel repetitions is not known to work in general, and in particular is not known to work in this specific case.

> **Author's Note:** *Again, the last statement is out of date and recent results do allow to reduce the error probability without increasing the number of rounds.*

6.11.4 Applications

Multi-prover interactive proofs are useful only in settings in which the "proving entity" can be separated and its parts kept ignorant of one another during the proving process. In such cases we get perfect zero-knowledge proofs without having to rely on complexity theoretic assumptions. In other words, general widely believed mathematical assumptions are replaced by physical assumptions concerning the specific setting.

A natural application is to the problem of identification, and specifically the identification of a *user* at some *station*. In Section 6.7 we discuss how to reduce identification to a zero-knowledge proof of knowledge (for some NP relation). The idea is to supply each user with two smart-cards, implementing the two provers in a two-prover zero-knowledge

proof of knowledge. These two smart-cards have to be inserted in two different slots of the station, and this guarantees that the smart-cards cannot communicate one with another. The station will play the role of the verifier in the zero-knowledge proof of knowledge. This way the station is protected against impersonation, whereas the users are protected against pirate stations which may try to extract knowledge from the smart-cards (so to enable impersonation by its agents).

6.12 Miscellaneous

6.12.1 Historical Notes

Interactive proof systems were introduced by Goldwasser, Micali and Rackoff [GMR85]. (Earlier versions of this paper date to early 1983. Yet, the paper, being rejected three times from major conferences, has first appeared in public only in 1985, concurrently to the paper of Babai [B85].) A restricted form of interactive proofs, known by the name *Arthur Merlin Games*, was introduced by Babai [B85]. (The restricted form turned out to be equivalent in power – see Section [missing(eff-ip.sec)].) The interactive proof for Graph Non-Isomorphism is due to Goldreich, Micali and Wigderson [GMW86].

The concept of zero-knowledge has been introduced by Goldwasser, Micali and Rackoff, in the same paper quoted above [GMR85]. Their paper contained also a perfect zero-knowledge proof for Quadratic Non Residuousity. The perfect zero-knowledge proof system for Graph Isomorphism is due to Goldreich, Micali and Wigderson [GMW86]. The latter paper is also the source to the zero-knowledge proof systems for all languages in $\mathcal{NP}$, using any (nonunifomly) one-way function. (Brassard and Crépeau have *later* constructed alternative zero-knowledge proof systems for $\mathcal{NP}$, using a *stronger* intractability assumption, specifically the intractability of the Quadratic Residuousity Problem.)

The cryptographic applications of zero-knowledge proofs were the very motivation for their presentation in [GMR85]. Zero-knowledge proofs were applied to solve cryptographic problems in [FMRW85] and [CF85]. However, many more applications were possible once it was shown how to construct zero-knowledge proof systems for every language in $\mathcal{NP}$. In particular, general methodologies for the construction of cryptographic protocols have appeared in [GMW86,GMW87].

Credits for the advanced sections

The results providing upper bounds on the complexity of languages with perfect zero-knowledge proofs (i.e., Theorem 6.36) are from Fortnow [For87] and Aiello and Hastad [AH87]. The results indicating that one-way functions are necessary for non-trivial zero-knowledge are from Ostrovsky and Wigderson [OWistcs93]. The negative results con-

cerning parallel composition of zero-knowledge proof systems (i.e., Proposition 6.37 and Theorem 6.39) are from [GKr89b].

The notions of witness indistinguishability and witness hiding, were introduced and developed by Feige and Shamir [FSwitness].

Author's Note: *FSwitness has appeared in STOC90.*

The concept of proofs of knowledge originates from the paper of Goldwasser, Micali and Rackoff [GMR85]. First attempts to provide a definition to this concept appear in Fiat, Feige and Shamir [FFS87] and Tompa and Woll [TW87]. However, the definitions provided in both [FFS87,TW87] are not satisfactory. The issue of defining proofs of knowledge has been extensively investigated by Bellare and Goldreich [BGknow], and we follow their suggestions. The application of zero-knowledge proofs of knowledge to identification schemes was discovered by Feige, Fiat and Shamir [FFS87].

Computationally sound proof systems (i.e., arguments) were introduced by Brassard, Chaum, and Crépeau [BCC87]. Their paper also presents perfect zero-knowledge arguments for $\mathcal{NP}$ based on the intractability of factoring. Naor et. al. [NOVY92] showed how to construct perfect zero-knowledge arguments for $\mathcal{NP}$ based on any one-way permutation, and Construction 6.58 is taken from their paper. The polylogarithmic-communication argument system for $\mathcal{NP}$ (of Subsection 6.8.4) is due to Kilian [K92].

Author's Note: *NOVY92 has appeared in Crypto92, and K92 in STOC92.*

Author's Note: *Micali's model of CS-proofs was intended for the missing chapter on complexity theory.*

The round-efficient zero-knowledge proof systems for $\mathcal{NP}$, based on any clawfree collection, is taken from Goldreich and Kahan [GKa89]. The round-efficient zero-knowledge arguments for $\mathcal{NP}$, based on any one-way function, uses ideas of Feige and Shamir [FSconst] (yet, their original construction is different).

Author's Note: *NIZK credits: BFM and others*

Multi-prover interactive proofs were introduced by Ben-Or, Goldwasser, Kilian and Wigderson [BGKW88]. Their paper also presents a perfect zero-knowledge two-prover proof system for $\mathcal{NP}$. The perfect zero-knowledge two-prover proof for $\mathcal{NP}$, presented in Section 6.11, follows their ideas but explicitly states the properties of the two-sender commitment scheme in use. Consequently, we observe that this proof system *can* be applied in parallel to decease the error probability to a negligible one.

Author's Note: *This observation escaped Feige, Lapidot and Shamir.*

6.12.2 Suggestion for Further Reading

For further details on interactive proof systems see Section [missing(eff-ip.sec)].

A uniform-complexity treatment of zero-knowledge was given by Goldreich [Guniform]. In particular, it is shown how to use (uniformly) one-way functions to construct interactive proof systems for $\mathcal{NP}$ so that it is infeasible to find instances on which the prover leaks knowledge.

Zero-knowledge proof systems for any language in $\mathcal{IP}$, based on (nonuniformly) one-way functions, were constructed by Impagliazzo and Yung [IY87] (yet, their paper contains no details). An alternative construction is presented by Ben-Or et. al. [Betal88].

Further reading related to the advanced sections

Additional negative results concerning zero-knowledge proofs of restricted types appear in Goldreich and Oren [GO87]. The interested reader is also directed to Boppana, Hastad and Zachos [BHZ87] for a proof that if every language in co$\mathcal{NP}$ has a constant-round interactive proof system then the Polynomial-Time Hierarchy collapses to its second level.

Round-efficient *perfect* zero-knowledge arguments for $\mathcal{NP}$, based on the intractability of the Discrete Logarithm Problem, appears in a paper by Brassard, Crépeau and Yung [BCY]. A round-efficient *perfect* zero-knowledge proof system for Graph Isomorphism appears in a paper by Bellare, Micali and Ostrovsky [BMO89].

> **Author's Note:** *NIZK suggestions*

An extremely efficient perfect zero-knowledge two-prover proof system for $\mathcal{NP}$, appears in a paper by Dwork et. al. [DFKNS]. Specifically, only logarithmic randomness and communication complexities are required to get a constant error probability. This result uses the characterization of $\mathcal{NP}$ in terms of low complexity multi-prover interactive proof systems, which is further discussed in Section [missing(eff-pcp.sec)].

The paper by Goldwasser, Micali and Rackoff [GMR85] contains also a suggestion for a general measure of "knowledge" revealed by a prover, of which zero-knowledge is merely a special case. For further details see Goldreich and Petrank [GPkc].

> **Author's Note:** *GPkc has appeared in FOCS91. See also a recent work by Goldreich, Ostrovsky and Petrank in STOC94.*

> **Author's Note:** *The discussion of knowledge complexity is better fit into the missing chapter on complexity.*

6.12.3 Open Problems

Our formulations of zero-knowledge (e.g., perfect zero-knowledge as defined in Definition 6.11) is different from the standard definition used in the literature (e.g., Definition 6.15). The standard definition refers to *expected* polynomial-time machines rather to strictly (probabilistic) polynomial-time machines. Clearly, Definition 6.11 implies Definition 6.15 (see Exercise 8), but it is open whether the converse hold.

Author's Note: *Base nizk and arguments on (more) general assumptions.*

6.12.4 Exercises

Exercise 1: *decreasing the error probability in interactive proofs*:
Prove Proposition 6.7.
(Guideline: Execute the weaker interactive proof sufficiently many times, using independently chosen coin tosses for each execution, and rule by an appropriate threshold. Observe that the bounds on completeness and soundness need to be efficiently computable. Be careful when demonstrating the soundness of the resulting verifier. The statement remains valid regardless of whether these repetitions are executed sequentially or "in parallel", yet demonstrating that the soundness condition is satisfied is much easier in the first case.)

Exercise 2: *the role of randomization in interactive proofs – part 1*: Prove that if L has an interactive proof system in which the verifier is deterministic then $L \in \mathcal{NP}$.
(Guideline: Note that if the verifier is deterministic then the entire interaction between the prover and the verifier is determined by the prover. Hence, a modified prover can just precompute the interaction and send it to the modified verifier as the only message. The modified verifier checks that the interaction is consistent with the message that the original verifier would have sent)

Exercise 3: *the role of randomization in interactive proofs – part 2*: Prove that if L has an interactive proof system then it has one in which the prover is deterministic. Furthermore, prove that for every (probabilistic) interactive machine V there exists a deterministic interactive machine P so that for every x the probability $\mathrm{Prob}(\langle P, V\rangle(x)=1)$ equals the supremum of $\mathrm{Prob}(\langle B, V\rangle(x)=1)$ taken over all interactive machines B.
(Guideline: for each possible prefix of interaction, the prover can determine a message which maximizes the accepting probability of the verifier V.)

Exercise 4: *the role of randomization in interactive proofs – part 3*: Consider a modification, to the definition of an interactive machine, in which the random-tapes of the prover and verifier coincide (i.e., intuitively, both use the same sequence of coin tosses which is known to both of them). Prove that every language having such a modified

interactive proof system has also an interactive proof system (of the original kind) in which the prover sends a single message.

Exercise 5: *the role of error in interactive proofs*: Prove that if L has an interactive proof system in which the verifier never (not even with negligible probability) accepts a string not in the language L then $L \in \mathcal{NP}$.
(Guideline: Define a relation R_L such that $(x,y) \in R_L$ if y is a full transcript of an interaction leading the verifier to accept the input x. We stress that y contains the verifier's coin tosses and all the messages received from the prover.)

Exercise 6: *error in perfect zero-knowledge simulators - part 1*: Consider modifications of Definition 6.11 in which condition 1 is replaced by requiring, for some function $q(\cdot)$, that $\text{Prob}(M^*(x) = \perp) < q(|x|)$. Assume that $q(\cdot)$ is polynomial-time computable. Show that if for some polynomials, $p_1(\cdot)$ and $p_2(\cdot)$, and all sufficiently large n's, $q(n) > 1/p_1(n)$ and $q(n) < 1 - 2^{-p_2(n)}$ then the modified definition is equivalent to the original one. Justify the bounds placed on the function $q(\cdot)$.
(Guideline: the idea is to repeatedly execute the simulator sufficiently many time.)

Exercise 7: *error in perfect zero-knowledge simulators - part 2*: Consider the following alternative to Definition 6.11, by which we say that (P,V) is *perfect zero-knowledge* if for every probabilistic polynomial-time interactive machine V^* there exists a probabilistic polynomial-time algorithm M^* so that the following two ensembles are statistically close (i.e., their statistical difference is negligible as a function of $|x|$)

- $\{\langle P, V^* \rangle(x)\}_{x \in L}$
- $\{M^*(x)\}_{x \in L}$

Prove that Definition 6.11 implies the new definition.

Exercise 8: (E) *error in perfect zero-knowledge simulators - part 3*: Prove that Definition 6.11 implies Definition 6.15.

Exercise 9: *error in computational zero-knowledge simulators*: Consider an alternative to Definition 6.12, by which the simulator is allowed to output the symbol $\perp$ (with probability bounded above by, say, $\frac{1}{2}$) and its output distribution is considered conditioned on its not being $\perp$ (as done in Definition 6.11). Prove that this alternative definition is equivalent to the original one (i.e., to Definition 6.12).

Exercise 10: *alternative formulation of zero-knowledge - simulating the interaction*: Prove the equivalence of Definitions 6.12 and 6.13.

Exercise 11: Present a simple probabilistic polynomial-time algorithm which simulates the view of the interaction of the verifier described in Construction 6.16 with the prover defined there. The simulator, on input $x \in GI$, should have output which is distributed identically to $\text{view}_{V_{GI}}^{P_{GI}}(x)$.

Exercise 12: Prove that the existence of bit commitment schemes implies the existence of one-way functions.
(Guideline: following the notations of Definition 6.20, consider the mapping of (v, s, r) to the receiver's view $(r, \overline{m})$. Observe that by the unambiguity requirement range elements are very unlikely to have inverses with both possible values of v. The mapping is polynomial-time computable and an algorithm that inverts it, even with success probability that is not negligible, can be used to contradict the secrecy requirement.)

Exercise 13: Considering the commitment scheme of Construction 6.23, suggest a cheating sender that induces a receiver-view (of the commit phase) being both

- indistinguishable from the receiver-view in interactions with the prescribed sender;
- with very high probability, neither a possible 0-commitment nor a possible 1-commitment.

(Hint: the sender just replies with a uniformly chosen string.)

Exercise 14: *using Construction 6.23 as a commitment scheme in Construction 6.25*: Prove that when the commitment scheme of Construction 6.23 is used in the *G3C* protocol then resulting scheme remains zero-knowledge. Consider the modifications required to prove Claim 6.26.2.

Exercise 15: *more efficient zero-knowledge proofs for $\mathcal{NP}$*: Following is an outline for a constant-round zero-knowledge proof for the Hamiltonian Circuit Problem (HCP), with acceptance gap $\frac{1}{2}$ (between inputs inside and outside of the language).

- *Common Input*: a graph $G=(V,E)$;
- *Auxiliary Input* (to the prover): a permutation ψ, over V, representing the order of vertices along a Hamiltonian Circuit;
- Prover's first step: Generates a random isomorphic copy of G, denoted $G' = (V, E')$. (Let π denote the permutation between G and G'). For each pair $(i,j) \in V^2$, the prover sets $e_{i,j} = 1$ if $(i,j) \in E'$ and $e_{i,j} = 0$ otherwise. The prover computes a random commitment to each $e_{i,j}$. Namely, it uniformly chooses $s_{i,j} \in \{0,1\}^n$ and computes $c_{i,j} = C_{s_{i,j}}(e_{i,j})$. The prover sends all the $c_{i,j}$'s to the verifier;
- Verifier's first step: Uniformly selects $\sigma \in \{0,1\}$ and sends it to the prover;
- Prover's second step: Let σ be the message received from the verifier. If $\sigma = 1$ then the prover reveals all the $|V|^2$ commitments to the verifier (by revealing all $s_{i,j}$'s), and sends along also the permutation π. If $\sigma = 0$ then the prover reveals only $|V|$ commitments to the verifier, specifically those corresponding to the Hamiltonian circuit in G' (i.e., the prover sends $s_{\pi(1),\pi(\psi(2))}$, $s_{\pi(2),\pi(\psi(3))}$,...,$s_{\pi(n-1),\pi(\psi(n))}$, $s_{\pi(n),\pi(\psi(1))}$).

Complete the description of the above interactive proof, evaluate its acceptance probabilities, and provide a sketch of the proof of the zero-knowledge property (i.e., describe the simulator). If you are really serious provide a full proof of the zero-knowledge property.

Exercise 16: *strong reductions*: Let L_1 and L_2 be two languages in $\mathcal{NP}$, and let R_1 and R_2 be binary relations characterizing L_1 and L_2, respectively. We say that the relation R_1 is *Levin-reducible* to the relation R_2 if there exist two polynomial-time computable functions f and g such that the following two conditions hold.

1. *standard requirement*: $x \in L_1$ if and only if $f(x) \in L_2$.
2. *additional requirement*: For every $(x, w) \in R_1$, it holds that $(f(x), g(w)) \in R_2$.

We call the above reduction after Levin, who upon discovering, independently of Cook and Karp, the existence of $\mathcal{NP}$-complete problem, made a stronger definition of a reduction which implies the above. Prove the following statements

1. Let $L \in \mathcal{NP}$ and let L_R be the generic relation characterizing L (i.e., fix a non-deterministic machine M_L and let $(x, w) \in R_L$ if w is an accepting computation of M_L on input x). Let R_{SAT} be the standard relation characterizing SAT (i.e., $(x, w) \in R_{SAT}$ if w is a truth assignment satisfying the CNF formula x). Prove that R_L is Levin-reducible to R_{SAT}.
2. Let R_{SAT} be as above, and let R_{3SAT} be defined analogously for $3SAT$. Prove that R_{SAT} is Levin-reducible to R_{3SAT}.
3. Let R_{3SAT} be as above, and let R_{G3C} be the standard relation characterizing $G3C$ (i.e., $(x, w) \in R_{G3C}$ if w is a 3-coloring of the graph x). Prove that R_{3SAT} is Levin-reducible to R_{G3C}.
4. Levin-reductions are transitive.

Exercise 17: Prove the existence of a Karp-reduction of L to SAT that, when considered as a function, can be inverted in polynomial-time. Same for the reduction of SAT to $3SAT$ and the reduction of $3SAT$ to $G3C$. (In fact, the standard Karp-reductions have this property.)

Exercise 18: *applications of Theorem 6.29*: Assuming the existence of non-uniformly one-way functions, present solutions to the following cryptographic problems:

1. Suppose that party R received over a public channel a message encrypted using its own public-key encryption. Suppose that the message consists of two parts and party R wishes to reveal to everybody the first part of the message but not the second. Further suppose that the other parties want a proof that R indeed revealed the correct contents of the first part of its message.

2. Suppose that party S wishes to send party R a signature to a publicly known document so that only R gets the signature but everybody else can verify that such a signature was indeed sent by S. (We assume that all parties share a public channel.)
3. Suppose that party S wishes to send party R a commitment to a partially specified statement so that R remains oblivious of the unspecified part. For example, S may wish to commit itself to some standard offer while keeping the amount offered secret.

Exercise 19: *on knowledge tightness*: Evaluate the knowledge tightness of Construction 6.25, when applied logarithmically many times *in parallel*.

Exercise 20: *error reduction in computationally sound proofs – part 1*: Given a computationally sound proof (with error probability $\frac{1}{3}$) for a language L construct a computationally sound proof with negligible error probability (for L).

Exercise 21: *error reduction in computationally sound proofs – part 2*: Construct a computationally sound proof that has negligible error probability (i.e., smaller than $1/p(|x|)$ for every polynomial $p(\cdot)$ and sufficiently long inputs x) but when repeated sequentially $|x|$ times has error probability greater than $2^{-|x|}$. We refer to the error probability in the (computational) soundness condition.

Exercise 22: *commitment schemes – an impossibility result*: Prove that there exists no two-party protocol which simultaneously satisfies the perfect secrecy requirement of Definition 6.57 and the (information theoretic) unambiguity requirement of Definition 6.20.

Exercise 23: *alternative formulation of black-box zero-knowledge*: We say that a probabilistic polynomial-time oracle machine M is a black-box simulator *for the prover P and the language L* if for every (not necessarily uniform) polynomial-size circuit family $\{B_n\}_{n\in\mathbb{N}}$, the ensembles $\{\langle P, B_{|x|}\rangle(x)\}_{x\in L}$ and $\{M^{B_{|x|}}(x)\}_{x\in L}$ are indistinguishable by (non-uniform) polynomial-size circuits. Namely, for every polynomial-size circuit family $\{D_n\}_{n\in\mathbb{N}}$, every polynomial $p(\cdot)$, all sufficiently large n and $x \in \{0,1\}^n \cap L$,

$$|\mathrm{Prob}\left(D_n(\langle P, B_n\rangle(x))=1\right) - \mathrm{Prob}\left(D_n(M^{B_n}(x))=1\right)| < \frac{1}{p(n)}$$

Prove that the current formulation is equivalent to the one presented in Definition 6.38.

Exercise 24: Prove that the protocol presented in Construction 6.25 is indeed a black-box zero-knowledge proof system for $G3C$.
(Guideline: use the formulation presented above.)

Chapter 7

Cryptographic Protocols

Author's Note: *This chapter is a serious obstacle to any future attempt of completing this book.*

```
%Plan
\input{pt-motiv}% Motivation (Examples: voting, OT)
\input{pt-def}%%% Definition (of a protocol problem)
%................ (2 and more parties, w/without ``fairness'')
\input{pt-two}%%% Construction of two-party protocols
\input{pt-many}%% Construction of multi-party protocols
%................ in the private-channel model.
%................ Adapt to the ``computational model'' (no private channels)
\input{pt-misc}%% As usual: History, Reading, Open, Exercises
```

Chapter 8

* New Frontiers

Where is the area going?

That's always hard to predict,
but following are some recent and not so recent developments.

```
%Plan
\input{fr-eff}%% more stress on efficiency (from a theory perspective!)
\input{fr-sys}%% "System Problems" (key-mgmt, replay, etc.)
\input{fr-dyn}%% Dynamic adversaries (in multi-party protocls)
\input{fr-incr}%% Incremental Cryptography [BGG]
\input{fr-traf}%% Trafic Analysis [RS]
\input{fr-soft}%% Software Protection [G,O] (that's not really new...)
```

Chapter 9

The Effect of Cryptography on Complexity Theory

Cryptography had a fundamental effect on the development of complexity theory. Notions such as computational indistinguishability, pseudorandomness (in the sense discussed in previous chapters), interactive proofs and random self-reducibility were first introduced and developed with a cryptographic motivation. However, these notions turned out to influence the development of complexity theory as well, and were further developed within this broader theory. In this chapter we survey some of these developments which have their roots in cryptography and yet provide results which are no longer (directly) relevant to cryptography.

```
%Plan
\input{eff-rand}% Deterministic Simulation of Randomized Complexity Classes
%................ (simulations of random-AC0, BPP and RL)
\input{eff-ip}%%% The power of Interactive Proofs (coNP subset IP=PSPACE)
\input{eff-pcp}%% PCP and its applications to hardness of approximation
\input{eff-rsr}%% Random Self-Reducibility (DLP/QR, Permanent)
\input{eff-lear}% Learning
\input{eff-misc}% (as usual)
```

Chapter 10

* Related Topics

In this chapter we survey several unrelated topics which are related to cryptography in some way. For example, a natural problem which arises in light of the excessive use of randomness is how to extract almost perfect randomness from sources of weak randomness.

```
%Plan
\input{tp-sour}%% Weak sources of randomness
\input{tp-byz}%%% Byzantine Agreement
\input{tp-check}% Program Checking and Statistical Tests
\input{tp-misc}%% As usual: History, Reading, Open, Exercises
```

Appendix A

Annotated List of References (compiled Feb. 1989)

Author's Note: *The following list of annotated references was compiled by me more than five years ago. The list was intended to serve as an appendix to class notes for my course on "Foundations of Cryptography" given at the Technion in the Spring of 1989. Thus, a few pointers to lectures given in the course appear in the list.*

Author's Note: *By the way, copies of the above-mentioned class notes, written mostly by graduate students attending my course, can be requested from the publication officer of the Computer Science Department of the Technion, Haifa, Israel. Although I have a very poor opinion of these notes, I was surprised to learn that they have been used by several people. The only thing that I can say in favour of these notes is that they cover my entire (one-semester) course on "Foundations of Cryptography"; in particular, they contain material on encryption and signatures (which is most missing in the current fragments).*

Preface

The list of references is partitioned into two parts: Main References and Suggestions for Further Reading. The Main References consists of the list of papers that I have *extensively* used during the course. Other papers which I mentioned briefly may be found in the list of Suggestions for Further Reading. This second list also contains papers, reporting further developments, which I have not mentioned at all.

Clearly, my suggestions for further reading do not exhaust all interesting works done in the area. Some good works were omitted on purpose (usually when totally superseeded by others) and some were omitted by mistake. Also, no consistent policy was implemented in deciding which version of the work to cite. In most cases I used the reference which I had available on line (as updating all references would have taken too much time).

PART I : Main References

[BM88] Bellare, M., and S. Micali, "How to Sign Given any Trapdoor Function", *Proc. 20th STOC*, 1988.

Simplifies the construction used in [GMR84], using a weaker condition (i.e. the existence of trapdoor one-way permutations).
Readability: reasonable.

[BM82] Blum, M., and Micali, S., "How to Generate Cryptographically Strong Sequences of Pseudo-Random Bits", *SIAM Jour. on Computing*, Vol. 13, 1984, pp. 850-864. First version in *FOCS 1982*.

Presents a general method of constructing pseudorandom generators, and the first example of using it. Characterizes such generators as passing all (polynomial-time) prediction tests. Presents the notion of a "hard-core" predicate and the first proof of the existence of such predicate based on the existence of a particular one-way function (i.e. Discrete Logarithm Problem).
Readability: confusing in some places, but usually fine.

[GL89] Goldreich, O., and L.A. Levin, "A Hard-Core Predicate to any One-Way Function", *21st STOC*, 1989, pp. 25-32.

Shows that any "padded" one-way function $f(x,p) = f_0(x) \cdot p$, has a simple hard-core bit, the inner-product mod-2 of x and p.
Readability: STOC version is very elegant and laconic (Levin wrote it). These notes present a more detailed but cumbersome version.

[GMW86] Goldreich, O., S. Micali, and A. Wigderson, "Proofs that Yield Nothing But their Validity and a Methodology of Cryptographic Protocol Design", *Proc. of 27th Symp. on Foundation of Computer Science*, 1986, pp. 174-187. A full version appears as TR-544, Computer Science Dept., Technion, Haifa, Israel.

Demonstrates the generality and the wide applicability of zero-knowledge proofs. In particular, using any bit commitment scheme, it is shown how to construct a zero-knowledge proof for any language in $\mathcal{NP}$. Perfect zero-knowledge proofs are presented for Graph Isomorphism and its complement.

Readability: the full version is very detailed, sometimes to a point of exhausting the reader. A more elegant proof of the main result is sketched in [G89a].

[GMW87] Goldreich, O., S. Micali, and A. Wigderson, "How to Play any Mental Game", *19th STOC*, 1987. A more reasonable version is available from me.

Deals with the problem of cryptographic protocols in its full generality, showing how to automatically generate fault-tolerant protocols for computing any function (using any trapdoor one-way permutation).
Readability: STOC version is too hand-waving. These notes constitute a better source of information.

[GM82] Goldwasser, S., and S. Micali, "Probabilistic Encryption", *JCSS*, Vol. 28, No. 2, 1984, pp. 270-299. Previous version in *STOC 1982*.

Introduces the concept of polynomially indistinguishable probability distributions. Presents notions of secure encryption, demonstrating the inadequency of previous intuitions. Presents a general method for constructing such encryption schemes, and a first application of it. First use of the "hybrid" method.
Readability: Nice introduction. The technical part is somewhat messy.

[GMR85] Goldwasser, S., S. Micali, and C. Rackoff, "The Knowledge Complexity of Interactive Proof Systems", *SIAM J. on Comput.*, Vol. 18, No. 1, 1989, pp. 186-208. Previous version in *STOC 1985*.

Introduces the concepts of an interactive proof and a zero-knowledge proof. Presents the first (non-trivial) example of a zero-knowledge proof. First application of zero-knowledge to the design of cryptographic protocols.
Readability: good.

[GMR84] Goldwasser, S., S. Micali, and R.L. Rivest, "A Digital Signature Scheme Secure Against Adaptive Chosen Message Attacks", *SIAM J on Comput.*, Vol. 17, No. 2, 1988, pp. 281-308. Previous version in *FOCS 1984*.

Surveys and investigates definitions of unforgeable signatures. Presents the first signature scheme which is unforgeable in a very strong sense even under a chosen message attack.
Readability: excellent as an introduction to the problem. Don't read the construction, but rather refer to [BM88].

[Y82] Yao, A.C., "Theory and Applications of Trapdoor Functions", *Proc. of the 23rd IEEE Symp. on Foundation of Computer Science*, 1982, pp. 80-91.

Presents a general definition of polynomially indistinguishable probability distributions. Characterizes pseudorandom generators as passing all (polynomial-time) statistical tests. (This formulation is equivalent to passing all polynomial-time prediction tests.) Given any one-way permutation constructs a pseudorandom generator.
Readability: Most interesting statements are not stated explicitly. Furthermore, contains no proofs.

PART II : Suggestions for Further Reading

My suggestions for further reading are grouped under the following categories:

1. *General:* Papers which deal or relate several of the following categories.
2. *Hard Computational Problems:* Pointers to literature on seemingly hard computational problems (e.g. integer factorization) and to works relating different hardness criteria.
3. *Encryption:* Papers dealing with secure encryption schemes (in the strong sense defined in lecture 5B and 6).
4. *Pseudorandomness:* Papers dealing with the construction of pseudorandom generators, pseudorandom functions and permutations and their applications to cryptography and complexity theory.
5. *Signatures and Commitment Schemes:* Papers dealing with unforable signature schemes (as defined in lecture 10) and secure commitment schemes (mentioned in lecture 13).
6. *Interactive Proofs, Zero-Knowledge and Protocols:* In addition to papers with apparent relevance to cryptography this list contains also papers investigating the complexity theoretic aspects of interactive proofs and zero-knowledge.
7. *Additional Topics:* Pointers to works on software protection, computation with an untrusted oracle, protection against abuse of cryptographic systems, Byzantine Agreement, sources of randomness, and "cryptanalysis".
8. *Historical Background:* The current approach to Cryptography did not emerge "out of the blue". It originates in works that were not referenced in the previous categories (which include only material conforming with the definitions and concepts presented in the course). This category lists some of these pioneering works.

A.1 General

Much of the current research in cryptography focuses on reducing the existence of complex cryptographic primitives (such as the existence of unforgeable signature schemes) to simple complexity assumptions (such as the existence of one-way functions). A first work investigating the limitations of these reductions is [IR89], where a "gap" between tasks implying secret key exchange and tasks reducible to the existence of one-way functions is shown. The gap is in the sense that a reduction of the first task to the second would imply $\mathcal{P} \neq \mathcal{NP}$.

Many of the more complex results in cryptography (e.g. the existence of zero-knowledge interactive proofs for all languages in $\mathcal{NP}$) are stated and proved in terms of non-uniform complexity. As demonstrated throughout the course, this simplifies both the statements and their proofs. An attempt to treat secure encryption and zero-knowledge in uniform complexity measures is reported in [G89a]. In fact, the lectures on secure encryption are based on [G89a].

references

[G89a] Goldreich, O., "A Uniform-Complexity Treatment of Encryption and Zero-Knowledge", TR-568, Computer Science Dept., Technion, Haifa, Israel, 1989.

[IR89] Impagliazzo, R., and S. Rudich, "Limits on the Provable Consequences of One-Way Permutations", *21st STOC*, pp. 44-61, 1989.

A.2 Hard Computational Problems

2.1. Candidates for One-Way functions

Hard computational problems are the basis of cryptography. The existence of adequately hard problems (see lecture 2) is not known. The most popular candidates are from computational number theory: integer factorization (see [P82] for a survey of the best algorithms known), discrete logarithms in finite fields (see [O84] for a survey of the best algorithms known), and the logarithm problem for "Elliptic groups" (cf. [M85]). Additional suggestions are the decoding problem for random linear codes (see [GKL88] and [BMT78]) and *high density* subset-sum ("knapsack") problems (see [CR88, IN89]). Note that *low density* subset-sum problems are usually easy (see survey [BO88]).

Much of the early-80th research in cryptography used the intractability assumption of the Quadratic Residuousity Problem (introduced in [GM82]). The nice structure of the problem was relied upon in constructions as [LMR83], but in many cases further research led to getting rid of the need to rely on the special structure (and to using weaker intractability assumptions).

Attempts to base cryptography on computationally hard combinatorial problems have been less popular. Graph Isomorphism is very appealing (as it has a nice structure as the Quadratic Residuousity Problem), but such a suggestion should not be taken seriously unless one specifies an easily samplable instance distribution for which the problem seems hard.

For details on candidates whose conjectured hardness was refuted see category 7.6.

2.2. Generic Hard Problems

The universal one-way function presented in lecture 3 originates from [L85]. The same ideas were used in [G88a] and [AABFH88], but the context there is of "average case complexity" (originated in [L84] and surveyed in [G88a]). In this context "hard" means intractable on infinitely many instance lengths, rather than intractable on all but finitely many instance lengths. Such problems are less useful in cryptography.

2.3. Hard-Core Predicates

As pointed out in lecture 4, hard-core predicates are a useful tool in cryptography. Such predicates are known to exist for exponentiation modulo a prime [BM82], (more generally) for "repeated addition" in any Abelian group [K88] and for the RSA and Rabin (squaring mod N) functions [ACGS84]. Recall that the general result of [GL89] (see lectures 4-5A) guarantees the existence of hard-core predicates for any "padded" function.

references

[AABFH88] Abadi, M., E. Allender, A. Broder, J. Feigenbaum, and L. Hemachandra, "On Generating Hard, Solved Instances of Computational Problem", *Crypto88* proceedings.

[ACGS84] W. Alexi, B. Chor, O. Goldreich and C.P. Schnorr, "RSA and Rabin Functions: Certain Parts Are As Hard As the Whole", *SIAM Jour. on Computing*, Vol. 17, 1988, pp. 194-209. A preliminary version appeared in *Proc. 25th FOCS*, 1984, pp. 449-457.

[BM82] see main references.

[BMT78] Berlekamp, E.R., R.J. McEliece, and H.C.A. van Tilborg, "On the Inherent Intractability of Certain Coding Problems", *IEEE Trans. on Inform. Theory*, 1978.

[BO88] Brickell, E.F., and A.M. Odlyzko, "Cryptanalysis: A Survey of Recent Results", *Proceedings of the IEEE*, Vol. 76, pp. 578-593, 1988.

[CR88] Chor, B., and R.L. Rivest, "A Knapsack Type Public-Key Cryptosystem Based on Arithmetic in Finite Fields", *IEEE Trans. on Inf. Th.*, Vol. 34, pp. 901-909, 1988.

[G88a] Goldreich, O., "Towards a Theory of Average Case Complexity (a survey)", TR-531, Computer Science Dept., Technion, Haifa, Israel, 1988.

[GKL88] see category 4.

[GL89] see main references.

[GM82] see main references.

[IN89] Impagliazzo, R., and M. Naor, "Efficient Cryptographic Schemes Provable as Secure as Subset Sum", manuscript, 1989.

[K88] B.S. Kaliski, Jr., "Elliptic Curves and Cryptography: A Pseudorandom Bit Generator and Other Tools", Ph.D. Thesis, LCS, MIT, 1988.

[L84] Levin, L.A., "Average Case Complete Problems", *SIAM Jour. of Computing*, 1986, Vol. 15, pp. 285-286. Extended abstract in *16th STOC*, 1984.

[L85] see category 4.

[LW] D.L. Long and A. Wigderson, "How Discreet is Discrete Log?", *Proc. 15th STOC*, 1983, pp. 413-420. A better version ?

[LMR83] see category 6.

[M85] Miller, V.S., "Use of Elliptic Curves in Cryptography", *Crypto85 - Proceedings*, Lecture Notes in Computer Science, Vol. 218, Springer Verlag, 1985, pp. 417-426.

[O84] Odlyzko, A.M., "Discrete Logarithms in Finite Fields and their Cryptographic Significance", *Eurocrypt84* proceedings, Springer-Verlag, Lecture Notes in Computer Science, Vol. 209, pp. 224-314, 1985. manuscript.

[P82] Pomerance, C., "Analysis and Comparison of some Integer Factorization Algorithms", *Computational Methods in Number Theory:* Part I, H.W. Lenstra Jr. and R. Tijdeman eds., Math. Center Amsterdam, 1982, pp. 89-139.

A.3 Encryption

The efficient construction of a secure public-key encryption scheme, presented in lecture 8, originates from [BG84]. The *security* of this scheme is based on the intractability assumption of factoring, while its *efficiency* is comparable with that of the RSA. More generally, the scheme can be based on any trapdoor one-way permutation.

Non-uniform versions of the two definitions of security (presented in lecture 6) were shown equivalent in [MRS88]. These versions were also shown equivalent to a third definition appearing in [Y82].

The robustness of encryption schemes against active adversaries was addressed in [GMT82]. Folklore states that secret communication can be achieved over a channel controlled by an active adversary by use of bi-directional communication: for every message transmission, the communicating parties exchange new authenticated cryptographic keys (i.e. the receiver transmits a new authenticated encryption-key that is used only for the current message). Note that this prevents a chosen message attack on the currently used instance of the encryption scheme. Note that this suggestion does not constitute a public-key encryption scheme, but rather a secure means of private bi-directional communication. It was claimed that "non-interactive zero-knowledge proofs of knowledge" yield the construction of public-key encryption secure against chosen ciphertext attack [BFM88], but no proof of this claim has appeared.

references

[BFM88] see category 6.

[BG84] Blum, M., and S. Goldwasser, "An Efficient Probabilistic Public-Key Encryption Scheme which hides all partial information", *Advances in Cryptology: Proc. of Crypto 84*, ed. B Blakely, Springer Verlag Lecture Notes in Computer Science, vol. 196, pp. 289-302.

[GMT82] Goldwasser, S., S. Micali, and P. Tong, "Why and How to Establish a Private Code in a Public Network", *23rd FOCS*, 1982, pp. 134-144.

[MRS88] Micali, S., C. Rackoff, and B. Sloan, "The Notion of Security for Probabilistic Cryptosystems", *SIAM Jour. of Computing*, 1988, Vol. 17, pp. 412-426.

[Y82] see main references.

A.4 Pseudorandomness

I have partitioned the works in this category into two subcategories: works with immediate cryptographic relevance versus works which have a more abstract (say complexity theoretic) orientation. A survey on Pseudorandomness in contained in [G88b].

4.1. Cryptographically oriented works

The theory of pseudorandomness was extended to deal with functions and permutations. Definitions of pseudorandom functions and permutations are presented in [GGM84] and [LR86]. Pseudorandom generators were used to construct pseudorandom functions [GGM84], and these were used to construct pseudorandom permutations [LR86]. Cryptographic applications are discussed in [GGM84b, LR86].

In lecture 9, we proved that the existence of one-way *permutations* implies the existence of pseudorandom generators. Recently, it has been shown that pseudorandom generators exist if and only if one-way functions exist [ILL89, H89]. The construction of pseudorandom generators presented in these works is very complex and inefficient, thus the quest for an efficient construction of pseudorandom generator based on any one-way function is not over yet. A previous construction by [GKL88] might turn out useful in this quest.

A very efficient pseudorandom generator based on the intractability of factoring integers arises from the works [BBS82, ACGS84, VV84]. The generator was suggested in [BBS82] (where it was proved secure assuming intractability of Quadratic Residuousity Problem), and proven secure assuming intractability of factoring in [VV84] (by adapting the techniques in [ACGS84]).

4.2. Complexity oriented works

The existence of a pseudorandom generator implies the existence of a pair of statistically different efficiently constructible probability ensembles which are computationally indistinguishable. This sufficient condition turns out to be also a necessary one [G89b].

The difference between the output distribution of a pseudorandom generator and more commonly considered distributions is demonstrated in [L88]. The "commonly considered" distributions (e.g. all distributions having a polynomial-time computable distribution function) are shown to be *homogenous* while a pseudorandom generator gives rise to distributions which are not homogenous. Homogenous distributions are defined as distributions which allow good average approximation of all polynomial-time invariant characteristics of a string from its Kolmogorov complexity.

The use of pseudorandom generators for deterministic simulation of probabilistic complexity classes was first suggested in [Y82]. A unified approach, leading to better simulations, can be found in [NW88]. Other results concerning the "efficient" generation of sequences which "look random" to machines of various complexity classes can be found in [RT85, BNS89, Ni89].

The existence of sparse and evasive pseudorandom distributions is investigated in [GKr89a]. A sparse distribution (unlike a distribution statistically close to the uniform one) ranges over a negligible fraction of the strings. Evasiveness is the infeasibility of hitting an element in the distribution's support. Applications of some results to zero-knowledge are presented in [GKr89b].

references

[ACGS84] see category 2.

[BNS89] Babai, L., N. Nisan, and M. Szegedy, "Multi-party Protocols and Logspace-Hard Pseudorandom Sequences", *21st STOC*, pp. 1-11, 1989.

[BBS82] L. Blum, M. Blum and M. Shub, *A Simple Secure Unpredictable Pseudo-Random Number Generator*, *SIAM Jour. on Computing*, Vol. 15, 1986, pp. 364-383. Preliminary version in *Crypto82*.

[BM82] see main references.

[G88b] Goldreich, O., "Randomness, Interactive Proofs, and Zero-Knowledge - A Survey", *The Universal Turing Machine - A Half-Century Survey*, R. Herken ed., Oxford Science Publications, pp. 377-406, 1988.

[G89b] Goldreich, O., "A Note on Computational Indistinguishability", TR-89-051, ICSI, Berkeley, USA, (1989).

[GGM84] Goldreich, O., S. Goldwasser, and S. Micali, "How to Construct Random Functions", *Jour. of ACM*, Vol. 33, No. 4, 1986, pp. 792-807. Extended abstract in *FOCS84*.

[GGM84b] Goldreich, O., S. Goldwasser, and S. Micali, "On the Cryptographic Applications of Random Functions", *Crypto84*, proceedings, Springer-Verlag, Lecture Notes in Computer Science, vol. 196, pp. 276-288, 1985.

[GKr89a] Goldreich, O., and H. Krawczyk, "Sparse Pseudorandom Distributions", *Crypto89* proceedings, to appear.

[GKr89b] see category 6.

[GKL88] Goldreich, O., H. Krawczyk, and M. Luby, "On the Existence of Pseudorandom Generators", *29th FOCS*, 1988.

[GM82] see main references.

[H89] Hastad, J., "Pseudo-Random Generators with Uniform Assumptions", preprint, 1989.

[ILL89] Impagliazzo, R., L.A. Levin, and M. Luby, "Pseudorandom Generation from One-Way Functions", *21st STOC*, pp. 12-24, 1989.

[L85] L.A. Levin, "One-Way Function and Pseudorandom Generators", *Combinatorica*, Vol. 7, No. 4, 1987, pp. 357-363. A preliminary version appeared in *Proc. 17th STOC*, 1985, pp. 363-365.

[L88] L.A. Levin, "Homogenous Measures and Polynomial Time Invariants", *29th FOCS*, pp. 36-41, 1988.

[LR86] M. Luby and C. Rackoff, "How to Construct Pseudorandom Permutations From Pseudorandom Functions", *SIAM Jour. on Computing*, Vol. 17, 1988, pp. 373-386. Extended abstract in *FOCS86.*

[NW88] Nisan, N., and A. Wigderson, "Hardness vs. Randomness", *Proc. 29th FOCS*, pp. 2-11, 1988.

[Ni89] Nisan, N., "Pseudorandom Generators for Bounded Space Machines", private communication, 1989.

[RT85] Reif, J.H., and J.D. Tygar, "Efficient Parallel Pseudo-Random Number Generation", *Crypto85*, proceedings, Springer-Verlag, Lecture Notes in Computer Science, vol. 218, pp. 433-446, 1985.

[Y82] see main references.

[VV84] Vazirani, U.V., and V.V. Vazirani, "Efficient and Secure Pseudo-Random Number Generation", *25th FOCS*, pp. 458-463, 1984.

A.5 Signatures and Commitment Schemes

Recent works reduce the existence of these important primitives to assumptions weaker than ever conjectured.

5.1. Unforgeable Signatures Schemes

Unforgeable signature schemes can be constructed assuming the existence of one-way permutations [NY89]. The core of this work is a method for constructing "cryptographically strong" hashing functions. Further improvements and techniques are reported in [G86, EGM89]: in [G86] a technique for making schemes as [GMR84, BM88, NY89] "memoryless" is presented; in [EGS89] the concept of "on-line/off-line" signature schemes is presented and methods for constructing such schemes are presented as well.

5.2. Secure Commitment Schemes

Secure commitment schemes can be constructed assuming the existence of pseudorandom generator [N89]. In fact, the second scheme presented in lecture 13 originates from this paper.

references

[BM88] see main references.

[EGM89] Even, S., O. Goldreich, and S. Micali, "On-Line/Off-Line Digital Signature Schemes", *Crypto89* proceedings, to appear.

[G86] Goldreich, O., "Two Remarks concerning the Goldwasser-Micali-Rivest Signature Scheme", *Crypto86*, proceedings, Springer-Verlag, Lecture Notes in Computer Science, vol. 263, pp. 104-110, 1987.

[GMR84] see main references.

[N89] M. Naor, "Bit Commitment Using Pseudorandomness", IBM research report. Also to appear in *Crypto89* proceedings, 1989.

[NY89] M. Naor and M. Yung, "Universal One-Way Hash Functions and their Cryptographic Applications", *21st STOC*, pp. 33-43, 1989.

A.6 Interactive Proofs, Zero-Knowledge and Protocols

This category is subdivided into three parts. The first contains mainly cryptographically oriented works on zero-knowledge, the second contains more complexity oriented works on interactive proofs and zero-knowledge. The third subcategory lists works on the design of

cryptographic protocols. Surveys on Interactive Proof Systems and Zero-Knowledge Proofs can be found in [G88b, Gw89].

6.1. Cryptographically oriented works on Zero-Knowledge

An important measure for the "practicality" of a zero-knowledge proof system is its *knowledge tightness*. Intuitively, tightness is (the supremum taken over all probabilistic polynomial-time verifiers of) the ratio between the time it takes the simulator to simulate an interaction with the prover and the complexity of the corresponding verifier [G87a]. The definition of zero-knowledge only guarantees that the knowledge-tightness can be bounded by any function growing faster than every polynomial. However, the definition does not guarantee that the knowledge-tightness can be bounded above by a *particular* polynomial. It is easy to see that the knowledge-tightness of the proof system for Graph Isomorphism (presented in lecture 12) is 2, while the tightness of proof system for Graph colouring (lecture 13) is m (i.e., the number of edges). I believe that the knowledge-tightness of a protocol is an important aspect to be considered, and that it is very desirable to have tightness be a constant. Furthermore, using the notion of knowledge-tightness one can introduce more refined notions of zero-knowledge and in particular the notion of constant-tightness zero-knowledge. Such refined notions may be applied in a non-trivial manner also to languages in $\mathcal{P}$.

Two standard efficiency measures associated with interactive proof systems are the *computational complexity* of the proof system (i.e., number of steps taken by either or both parties) and the *communication complexity* of the proof system (here one may consider the number of rounds, and/or the total number of bits exchanged). Of special importance to practice is the question whether the (honest) prover's program can be a probabilistic polynomial-time when an auxiliary input is given (as in the case of the proof system, presented in lecture 13, for Graph Colouability). An additional measure, the importance of which has been realized only recently, is the number of strings to which the commitment scheme is applied individually (see [KMO89]). The zero-knowledge proof system for graph colourability presented in lecture 13 is not the most practical one known. Proof systems with constant knowledge-tightness, probabilistic polynomial-time provers and a number of iterations which is merely super-logarithmic exist for all languages in $\mathcal{NP}$ (assuming, of course, the existence of secure commitment) [IY87]. This proof system can be modified to yield a zero-knowledge proof with $f(n)$ iterations, for every unbounded function f. Using stronger intractability assumptions (e.g. the existence of claw-free one-way permutations), constant-round zero-knowledge proof systems can be presented for every language in $\mathcal{NP}$ [GKa89].

Perfect zero-knowledge *arguments*[1] were introduced in [BC86a, BCC88] and shown to exist for all languages in $\mathcal{NP}$, assuming the intractability of factoring integers. The differ-

[1] The term "argument" has appeared first in [BCY89]. The authors of [BCC88] create an enormous

ence between arguments and interactive proofs is that in a argument the soundness condition is restricted to probabilistic polynomial-time machines (with auxiliary input). Hence, it is *infeasible* (not impossible) to fool the verifier into accepting (with non-negligible probability) an input not in the language. Assuming the existence of any commitment scheme, it is shown that any language in $\mathcal{NP}$ has a constant-round zero-knowledge argument [FS88].

The limitations of zero-knowledge proof systems and the techniques to demonstrate their existence are investigated in [GO87, GKr89b]. In particular, zero-knowledge proofs with deterministic verifier (resp. prover) exist only for languages in $\mathcal{RP}$ (resp. $\mathcal{BPP}$), constant-round proofs of the AM-type (cf. [B85]) can be demonstrated zero-knowledge by an oblivious simulation only if the language is in $\mathcal{BPP}$. Thus, the "parallel versions" of the interactive proofs (presented in [GMW86]) for Graph Isomorphism and every $L \in \mathcal{NP}$ are unlikely to be demonstrated zero-knowledge. However, modified versions of these interactive proofs yield constant-round zero-knowledge proofs (see [GKa89] for $\mathcal{NP}$ and [BMO89] for Graph Isomorphism). These interactive proofs are, of course, not of the AM-type.

The concept of a "proof of knowledge" was introduced and informally defined in [GMR85]. Precise formalizations following this sketch has appeared in [BCC88, FFS87, TW87]. This concept is quite useful in the design of cryptographic protocols and zero-knowledge proof systems. In fact, it has been used implicitly in [GMR85, GMW87, CR87] and explicitly in [FFS87, TW87]. However, I am not too happy with the current formalizations and intend to present a new formalization.

"Non-interactive" zero-knowledge proofs are known to exist assuming the existence of trapdoor one-way permutations [KMO89]. These are two-phase protocols. The first phase is a preprocessing which uses bi-directional communication. In the second phase, zero-knowledge proofs can be produced via one-directional communication from the prover to the verifier. The number of statements proven in the second phase is a polynomial in the complexity of the first phase (this polynomial is arbitrarily fixed *after* the first phase is completed).

Historical remark: Using a stronger intractability assumption (i.e. the intractability of Quadratic Residuousity Problem) [BC86b] showed that every language in $\mathcal{NP}$ has a zero-knowledge interactive proof system. This result has been obtained independently of (but subsequently to) [GMW86].

---

amount of confusion by insisting to refer to arguments by the term interactive proofs. For example, the result of [For87] does not hold for perfect zero-knowledge *arguments*. Be careful not to confuse arguments with interactive proofs in which the completeness condition is satisfied by a probabilistic polynomial-time prover (with auxiliary input).

6.2. Complexity oriented works on Interactive Proofs and Zero-Knowledge

The definition of interactive proof systems, presented in lecture 12, originates from [GMR85]. A special case, in which the verifier sends the outcome of all its coin tosses to the prover was suggested in [B85] and termed *Arthur Merlin* (AM) games. AM games are easier to analyze, while general interactive proof systems are easier to design. Fortunately, the two formalizations coincide in a strong sense: for every polynomial Q, the classes $\mathcal{IP}(Q(n))$ and $\mathcal{AM}(Q(n))$ are equal [GS86], where $\mathcal{IP}(Q(n))$ denotes the class of languages having $Q(n)$-round interactive proof system. It is also known, that for every $k \geq 1$ and every polynomial Q, the class $\mathcal{AM}(Q(n))$ and $\mathcal{AM}(k \cdot Q(n))$ coincide [BaMo88]. A stronger result does not "relativize" (i.e. there exists an oracle A such that for every polynomial Q and every unbounded function g the class $\mathcal{AM}(Q(n))^A$ is strictly contained in $\mathcal{AM}(g(n) \cdot Q(n))^A$) [AGH88].

> **Author's Note:** *However, in light of the results of* [LFKN,S] *(see* FOCS90*), this means even less than ever. See also Chang et. al. (*JCSS*, Vol. 49, No. 1).*

> **Author's Note:** *This list was compiled before the fundamental results of Lund, Fortnow, Karloff and Nisan* [LFKN] *and Shamir* [S] *were known. By these results every language in $\mathcal{PSPACE}$ has an interactive proof system. Since $\mathcal{IP} \subseteq \mathcal{PSPACE}$ [folklore], the two classes collide.*

Every language $L \in \mathcal{IP}(Q(n))$ has a $Q(n)$-round interactive proof system in which the verifier accepts every $x \in L$ with probability 1, but only languages in $\mathcal{NP}$ have interactive proof systems in which the verifier never accepts $x \notin L$ [GMS87]. Further developments appear in [BMO89].

The class $\mathcal{AM}(2)$ is unlikely to contain co$\mathcal{NP}$, as this will imply the collapse of the polynomial-time hierarchy [BHZ87]. It is also known that for a random oracle A, $\mathcal{AM}(2) = \mathcal{NP}^A$ [NW88].

The complexity of languages having zero-knowledge proof systems seems to depend on whether these systems are *perfect* or only *computational* zero-knowledge. On one hand, it is known that perfect (even almost-perfect) zero-knowledge proof systems exist only for languages inside $\mathcal{AM}(2) \cap \text{co}\mathcal{AM}(2)$ [For87, AH87]. On the other hand, assuming the existence of commitment schemes (the very assumption used to show "NP in ZK") every languages in $\mathcal{IP}$ has a computational zero-knowledge proof system [IY87] (for a detailed proof see [Betal88]). Returning to perfect zero-knowledge proof systems, it is worthwhile mentioning that such systems are known for several computational problems which are considered hard (e.g. Quadratic Residuousity Problem [GMR85], Graph Isomorphism [GMW86], membership in a subgroup [TW87], and a problem computationally equivalent to Discrete Logarithm [GKu88]).

The concept of the knowledge complexity of a languages was introduced in [GMR85], but the particular formalization suggested there is somewhat ad-hoc and unnatural.[2] The *knowledge complexity* of a language is the minimum number of bits released by an interactive proof system for the language. Namely, a language $L \in \mathcal{IP}$ has knowledge complexity $\leq k(\cdot)$ if there exists an interactive proof for L such that the interaction of the prover on $x \in L$ can be simulated by a probabilistic polynomial-time *oracle* machine on input x and up to $k(|x|)$ Boolean queries (to an oracle of "its choice"). More details will appear in a forthcoming paper of mine.

An attempt to get rid of the intractability assumption used in the "NP in ZK" result of [GMW86], led [BGKW88] to suggest and investigate a model of *multi-prover* interactive proof systems. It was shown that two "isolated" provers can prove statements in $\mathcal{NP}$ in a perfect zero-knowledge manner. A different multi-prover model, in which one unknown prover is honest while the rest my interact and cheat arbitrarily, was suggested and investigated in [FST88]. This model is equivalent to computation with a "noisy oracle".

6.3. On the Design of Cryptographic Protocols

The primary motivation for the concept of zero-knowledge proof systems has been their potential use in the design of cryptographic protocols. Early examples of such use can be found in [GMR85, FMRW85, CF85]. The general results in [GMW86] allowed the presentation of automatic generators of two-party and multi-party cryptographic protocols (see [Y86][3] and [GMW87], respectively). Further improvements are reported in [GHY87, GV87, IY87].

Two important tools in the construction of cryptographic protocols are Oblivious Transfer and Verifiable Secret Sharing. *Oblivious Transfer*, introduced in [R81], was further investigated in [EGL82, FMRW85, BCR86, Cre87, CK88, Kil88]. *Verifiable Secret Sharing*, introduced in [CGMA85], was further investigated in [GMW86, Bh86a, Fel87]. Other useful techniques appear in [Bh86b, CR87].

An elegant model for investigations of multi-party cryptographic protocols was suggested in [BGW88]. This model consists of processors connected in pairs via *private channels*. The bad processors have infinite computing resources (and so using computationally hard problems is useless). Hence, computational complexity restrictions and assumptions are substituted by assumptions about the communication model. An automatic generator of protocols for this model, tolerating up to $\frac{1}{3}$ malicious processors, has been presented in [BGW88, CCD88]. Augmenting the model by a broadcast channel, tolerance can be

[2] In particular, according to that formalization a prover revealing with probability $\frac{1}{2}$ a Hamiltonian circuit in the input gragh yields one one bit of knowledge.

[3] It should be stressed that [Y86] improves over [Y82b]. The earlier paper presented two-party cryptographic protocols allowing semi-honest parties to compute privately functions ranging over "small" (i.e. polynomially bounded) domains.

improved to $\frac{1}{2}$ [BR89]. (The augmentation is necessary, as there are tasks which cannot be performed if a third of the processors are malicious (e.g. Byzantine Agreement).) Beyond the $\frac{1}{2}$ bound, only functions of special type (i.e. the exclusive-or of locally computed functions) can be privately computed [CKu89].

references

[AGH86] Aiello, W., S. Goldwasser, and J. Hastad, "On the Power of Interaction", *Proc. 27th FOCS*, pp. 368-379, 1986.

[AH87] Aiello, W., and J. Hastad, "Perfect Zero-Knowledge Languages can be Recognized in Two Rounds", *Proc. 28th FOCS*, pp. 439-448, 1987.

[AGY85] Alon, N., Z. Galil, and M. Yung, "A Fully Polynomial Simultaneous Broadcast in the Presence of Faults", unpublished manuscript, 1985.

[B85] Babai, L., "Trading Group Theory for Randomness", *Proc. 17th STOC*, 1985, pp. 421-429.

[BKL] Babai, L., W.M. Kantor, and E.M. Luks, "Computational Complexity and Classification of Finite Simple Groups", *Proc. 24th FOCS*, pp. 162-171, 1983.

[BaMo88] Babai, L., and S. Moran, "Arthur-Merlin Games: A Randomized Proof System, and a Hierarchy of Complexity Classes", *JCSS*, Vol. 36, No. 2, pp. 254-276, 1988.

[BMO89] Bellare, M., S. Micali, and R. Ostrovsky, "On Parallelizing Zero-Knowledge Proofs and Perfect Completeness Zero-Knowledge", manuscript, April 1989.

[Bh86a] Benaloh, (Cohen), J.D., "Secret Sharing Homomorphisms: keeping shares of a secret secret", *Crypto86*, proceedings, Springer-Verlag, Lecture Notes in Computer Science, vol. 263, pp. 251-260, 1987.

[Bh86b] Benaloh, (Cohen), J.D., "Cryptographic Capsules: A Disjunctive Primitive for Interactive Protocols", *Crypto86*, proceedings, Springer-Verlag, Lecture Notes in Computer Science, vol. 263, pp. 213-222, 1987.

[Betal88] Ben-Or, M., O. Goldreich, S. Goldwasser, J. Hastad, J. Killian, S. Micali, and P. Rogaway, "Every Thing Provable is provable in ZK", to appear in the proceedings of *Crypto88*, 1988.

[BGW88] Ben-Or, M., S. Goldwasser, and A. Wigderson, "Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation", *20th STOC*, pp. 1-10, 1988.

[BGKW88] Ben-Or, M., S. Goldwasser, J. Kilian, and A. Wigderson, "Multi-Prover Interactive Proofs: How to Remove Intractability", *20th STOC*, pp. 113-131, 1988.

[BT89] Ben-Or, M., and T. Rabin, "Verifiable Secret Sharing and Multiparty Protocols with Honest Majority", *21st STOC*, pp. 73-85, 1989.

[Bk] Blakley, G.R., "Safeguarding Cryptographic Keys", *Proc. of National Computer Conf.*, Vol. 48, AFIPS Press, 1979, pp. 313-317.

[BFM88] Blum, M., P. Feldman, and S. Micali, "Non-Interactive Zero-Knowledge and its Applications", *20th STOC*, pp. 103-112, 1988.

[BHZ87] Boppana, R., J. Hastad, and S. Zachos, "Does Co-NP Have Short Interactive Proofs?", *IPL*, 25, May 1987, pp. 127-132.

[BCC88] Brassard, G., D. Chaum, and C. Crepeau, "Minimum Disclosure Proofs of knowledge", *JCSS*, Vol. 37, No. 2, Oct. 1988, pp. 156-189.

[BC86a] Brassard, G., and C. Crepeau, "Non-Transitive Transfer of Confidence: A Perfect Zero-Knowledge Interactive Protocol for SAT and Beyond", *Proc. 27th FOCS*, pp. 188-195, 1986.

[BC86b] Brassard, G., and C. Crepeau, "Zero-Knowledge Simulation of Boolean Circuits", *Advances in Cryptology - Crypto86 (proceedings)*, A.M. Odlyzko (ed.), Springer-Verlag, Lecture Notes in Computer Science, vol. 263, pp. 223-233, 1987.

[BCR86] Brassard, G., C. Crepeau, and J.M. Robert, "Information Theoretic Reductions Among Disclosure Problems", *Proc. 27th FOCS*, pp. 168-173, 1986.

[BCY89] Brassard, G., C. Crepeau, and M. Yung, "Everything in $\mathcal{NP}$ can be argued in perfect zero-knowledge in a bounded number of rounds", *Proc. of the 16th ICALP*, July 1989.

[CCD88] Chaum, D., C. Crepeau, I. Dangard, "Multi-party Unconditionally Secure Protocols", *20th STOC*, pp. 11-19, 1988.

[Cha] Chaum, D., "Demonstrating that a Public Predicate can be Satisfied Without Revealing Any Information About How", *Advances in Cryptology - Crypto86 (proceedings)*, A.M. Odlyzko (ed.), Springer-Verlag, Lecture Notes in Computer Science, vol. 263, pp. 195-199, 1987.

[CGMA85] Chor, B., S. Goldwasser, S. Micali, and B. Awerbuch, "Verifiable Secret Sharing and Achieving Simultaneity in the Presence of Faults", *Proc. 26th FOCS*, 1985, pp. 383-395.

[CKu89] Chor, B., and E. Kushilevitz, "A Zero-One Law for Boolean Privacy", *21st STOC*, pp. 62-72, 1989.

[CR87] Chor, B., and M.O, Rabin, "Achieving Independence in Logarithmic Number of Rounds", *6th PODC*, pp. 260-268, 1987.

[CGG] Chor, B., O. Goldreich, and S. Goldwasser, "The Bit Security of Modular Squaring given Partial Factorization of the Modulos", *Advances in Cryptology - Crypto85 (proceedings)*, H.C. Williams (ed.), Springer-Verlag, Lecture Notes in Computer Science, vol. 218, 1986, pp. 448-457.

[CF85] Cohen, J.D., and M.J. Fischer, "A Robust and Verifiable Cryptographically Secure Election Scheme", *Proc. 26th FOCS*, pp. 372-382, 1985.

[Cre87] Crepeau, C., "Equivalence between two Flavour of Oblivious Transfer", *Crypto87* proceedings, Lecture Notes in Computer Science, Vol. 293, Springer-Verlag, 1987, pp. 350-354.

[CK88] Crepeau, C., and J. Kilian, "Weakening Security Assumptions and Oblivious Transfer", *Crypto88* proceedings.

[EGL82] see category 8.

[Fel87] Feldman, P., "A Practical Scheme for Verifiable Secret Sharing", *Proc. 28th FOCS*, pp. 427-438, 1987.

[FFS87] Feige, U., A. Fiat, and A. Shamir, "Zero-Knowledge Proofs of Identity", *Proc. of 19th STOC*, pp. 210-217, 1987.

[FST88] Feige, U., A. Shamir, and M. Tennenholtz, "The Noisy Oracle Problem", *Crypto88* proceedings.

[FS88] Feige, U., and A. Shamir, "Zero-Knowledge Proofs of Knowledge in Two Rounds", manuscript, Nov. 1988.

[FMRW85] Fischer, M., S. Micali, C. Rackoff, and D.K. Wittenberg, "An Oblivious Transfer Protocol Equivalent to Factoring", unpublished manuscript, 1986. Preliminary versions were presented in *EuroCrypt84* (1984), and in the *NSF Workshop on Mathematical Theory of Security*, Endicott House (1985).

[For87] Fortnow, L., "The Complexity of Perfect Zero-Knowledge", *Proc. of 19th STOC*, pp. 204-209, 1987.

[GHY85] Galil, Z., S. Haber, and M. Yung, "A Private Interactive Test of a Boolean Predicate and Minimum-Knowledge Public-Key Cryptosystems", *Proc. 26th FOCS*, 1985, pp. 360-371.

[GHY87] Galil, Z., S. Haber, and M. Yung, "Cryptographic Computation: Secure Fault-Tolerant Protocols and the Public-Key Model" *Crypto87*, proceedings, Springer-Verlag, Lecture Notes in Computer Science, vol. 293, pp. 135-155, 1987.

[G87a] Goldreich, O., "Zero-Knowledge and the Design of Secure Protocols (an exposition)", TR-480, Computer Science Dept., Technion, Haifa, Israel, 1987.

[G88b] see category 4.

[GKu88] Goldreich, O., and E. Kushilevitz, "A Perfect Zero-Knowledge Proof for a Decision Problem Equivalent to Discrete Logarithm", *Crypto88*, proceedings.

[GKa89] Goldreich, O., and A. Kahan, "Using Claw-Free Permutations to Construct Zero-Knowledge Proofs for NP", in preparation, 1989.

[GKr89b] Goldreich, O., and H. Krawczyk, "On Sequential and Parallel Composition of Zero-Knowledge Protocols", preprint, 1989.

[GV87] Goldreich, O., and R. Vainish, "How to Solve any Protocol Problem - an Efficiency Improvement", *Crypto87*, proceedings, Springer-Verlag, Lecture Notes in Computer Science, vol. 293, pp. 73-86, 1987.

[GMS87] Goldreich, O., Y. Mansour, and M. Sipser "Interactive Proof Systems: Provers that Never Fail and Random Selection", *28th FOCS*, pp. 449-461, 1987.

[GMW86] see main references.

[GMW87] see main references.

[GO87] Goldreich, O., and Y. Oren, "On the Cunning Power of Cheating Verifiers: Some Observations about Zero-Knowledge Proofs", in preparation. Preliminary version, by Y. Oren, in *FOCS87*.

[Gw89] Goldwasser, S., "Interactive Proof Systems", *Proc. of Symposia in Applied Mathematics*, AMS, Vol. 38, 1989.

[GMR85] see main references.

[GS86] Goldwasser, S., and M. Sipser, "Private Coins vs. Public Coins in Interactive Proof Systems", *Proc. 18th STOC*, 1986, pp. 59-68.

[IY87] Impagliazzo, R., and M. Yung, "Direct Minimum-Knowledge Computations", *Advances in Cryptology - Crypto87 (proceedings)*, C. Pomerance (ed.), Springer-Verlag, Lecture Notes in Computer Science, vol. 293, 1987, pp. 40-51.

[Kil88] Kilian, J., "Founding Cryptography on Oblivious Transfer", *20th STOC*, pp. 20-31, 1988.

[LMR83] Luby, M., S. Micali, and C. Rackoff, *24th FOCS*, 1983.

[KMO89] Kilian, J., S. Micali, and R. Ostrovsky, "Simple Non-Interactive Zero-Knowledge Proofs", *30th FOCS*, to appear, 1989.

[NW88] see category 4.

[R81] see category 8.

[TW87] Tompa, M., and H. Woll, "Random Self-Reducibility and Zero-Knowledge Interactive Proofs of Possession of Information", *Proc. 28th FOCS*, pp. 472-482, 1987.

[Y82b] Yao, A.C., "Protocols for Secure Computations", *23rd FOCS*, 1982, pp. 160-164.

[Y86] Yao, A.C., "How to Generate and Exchange Secrets", *Proc. 27th FOCS*, pp. 162-167, 1986.

A.7 Additional Topics

This category provides pointers to topics which I did not address so far. These topics include additional cryptographic problems (e.g. software protection, computation with an untrusted oracle, and protection against "abuse of cryptographic systems"), lower level primitives (e.g. Byzantine Agreement and sources of randomness) and "cryptanalysis".

7.1. Software Protection

A theoretical framework for discussing software protection is suggested in [G87b]. Recently, the solution in [G87b] has been dramatically improved [O89].

7.2. Computation with an Untrusted Oracle

Computation with an untrusted oracle raises two problems: the oracle may fail the computation by providing wrong answers, and/or the oracle can gain information on the input of the machine which uses it. The first problem can be identified with recent research on "program checking" initiated in [BK89]. Note that the definition of "program checking" is more refined than the one of an interactive proof (in particular it does not trivialize polynomial-time computations and does not allow infinitely powerful provers) and thus is more suitable for the investigation. The results in [BK89, BLR89] are mainly encouraging as they provide many positive examples of computations which can be sped-up (and yet confirmed) using an oracle. A formalization of the second problem, presented in [AFK87], seems to have reached a dead-end with the negative results of [AFK87]. Other formalizations appear in [BF89] and [BLR89].

7.3. Protection Against Abuse of Cryptographic Systems

How can a third party prevent the abuse of a two-party cryptographic protocol executed through a channel he controls? As an example consider an attempt of one party to pass information to his counterpart by using a signature scheme. This old problem (sometimes referred to as *the prisoners' problem* or *the subliminal channel*) is formalized and solved, using active intervention of the third party, in [D88].

7.4. Byzantine Agreement

In lectures 14-15 we have assumed the existence of a *broadcast channel* accessable by all processors. In case such a channel does not exist in the network (i.e., in case we are using a point-to-point network), such a channel can be implemented using *Byzantine Agreement*. Using private channel, randomized Byzantine Agreement protocols with expected $O(1)$ rounds can be implemented [FM88]. This work builds on [R83]. Additional insight can be gained from the pioneering works of [Be83, Br85], and from the survey of [CD89].

7.5. Sources of Randomness

A subject related to cryptography is the use of weak sources of randomness in applications requiring perfect coins. Models of weak sources are presented and investigated in [B84, SV84, CG85, Cetal85, LLS87]. Further developments are reported in [V85, VV85, V87].

7.6. Cryptanalysis

In all the famous examples of successful cryptanalysis of a proposed cryptographic scheme, the success revealed a explicit or implicit assumption made by the designers of the cryptosystem. This should serve as experimental support to the thesis underlying the course that assumptions have to be made explicitly.

Knapsack cryptosystems, first suggested in [MH78], were the target of many attacks. The first dramatic success was the breaking of the original [MH78] scheme, using the existence of a trapdoor super-increasing sequence [S82]. An alternative attack applicable against *low density* knapsack (subset sum) problems was suggested in [LO85]. For more details see [BO88]. *It seems that the designers conjectured that subset sum problems with a trapdoor (resp. with low density) are as hard as random high density subset sum problems. It seems that this conjecture is false.*

Another target for many attacks were the linear congruential number generators and their generalizations. Although these generators are known to pass many statistical tests [K69], they do not pass all polynomial-time statistical tests [Boy82]. Generalizations to

polynomial congruential recurrences and linear generators which output only part of the bits of the numbers produced can be found in [Kr88] and [S87], respectively. *The fact that a proposed scheme passes some tests or attacks does not mean that it will pass all efficient tests.*

Another famous cryptographic system which triggered interesting algorithmic research is the [OSS84] signature scheme. This scheme was based on the conjecture, latter refuted in [Pol84], that it is hard to solve a modular quadratic equation in two variables. Other variants (e.g. [OSS84b, OS85]) were broken as well (in [EAKMM85, BD85], resp.). *Proving that one cannot find the trapdoor information used by the legal signer does not mean that one cannot forge signatures.*[4]

references

[AFK87] Abadi, M., J. Feigenbaum, and J. Kilian, "On Hiding Information from an Oracle", *19th STOC*, pp. 195-203, 1987.

[BF89] Beaver, D., and J. Feigenbaum, "Encrypted Queries to Multiple Oracles", manuscript, 1989.

[B84] Blum, M., "Independent Unbiased Coin Flips from a Correlated Biased Source: a Finite State Markov Chain", *25th Symp. on Foundation of Computer Science*, pp. 425-433, 1984.

[Be83] Ben-Or, M., "Another Advantage of Free Choice: Completely Asynchronous Agreement Protocols", *2nd PODC*, pp. 27-30, 1983.

[BK89] Blum, M., and S. Kannan, "Designing Programs that Check their Work", *21st STOC*, pp. 86-97, 1989.

[BLR89] Blum, M., M. Luby, and R. Rubinfeld, in preparation.

[Boy82] Boyar, J.B., "Inferring Sequences Produced by Pseudo-Random Number Generators", *JACM*, Vol. 36, No. 1, pp. 129-141, 1989. Early version in *FOCS82* (under previous name: Plumstead).

[Br85] Bracha, G., "An $O(\log n)$ Expected Rounds Randomized Byzantine Generals Protocol", *JACM*, Vol. 34, No. 4, pp. 910-920, 1987. Extended abstract in *STOC85*.

[4] To further stress this point, consider a signature scheme "based on composites" where the signature of a message m relative to the public-key N is $2m \bmod N$. The infeasibility of retrieving the trapdoor (i.e. the factorization of N) is a poor guarantee for security.

[BD85] Brickell, E.F., and J.M. DeLaurentis, "An Attack on a Signature Scheme Proposed by Okamoto and Shiraishi", *Crypto85*, proceedings, Springer-Verlag, Lecture Notes in Computer Science, vol. 218, pp. 28-32, 1985.

[LO85] Lagarias, J.C., and A.M. Odlyzko, "Solving Low-Density Subset Sum Problems", *JACM*, Vol. 32, (1985), pp. 229-246. *24th FOCS*, pp. 1-10, 1983.

[BO88] see category 2.

[CD89] Chor, B., and C. Dwork, "Randomization in Byzantine Agreement", *Advances in Computing Research*, Vol. 5, S. Micali, ed., JAI Press, in press.

[Cetal85] Chor, B., J. Freidmann, O. Goldreich, J. Hastad, S. Rudich, and R. Smolensky, "The Bit Extraction Problem or t-Resilient Functions", *26th FOCS*, pp. 396-407, 1985.

[CG85] Chor, B., and O. Goldreich, "Unbiased Bits from Sources of Weak Randomness and Probabilistic Communication Complexity", *26th Symp. on Foundation of Computer Science*, pp. 427-443, 1985.

[D88] Desmedt, Y., "Abuses in Cryptography and How to Fight Them", *Crypto88* proceedings, to appear.

[EAKMM85] Estes, D., L. Adleman, K. Kompella, K. McCurley, and G. Miller, "Breaking the Ong-Schnorr-Shamir Signature Scheme for Quadratic Number Fields", *Crypto85*, proceedings, Springer-Verlag, Lecture Notes in Computer Science, vol. 218, pp. 3-13, 1985.

[FM88] Feldman, P., and S. Micali, "Optimal Algorithms for Byzantine Agreement", *20th STOC*, pp. 148-161, 1988.

[FHKLS] Frieze, A.M., J. Hastad, R. Kannan, J.C. Lagarias, and A. Shamir, "Reconstructing Truncated Integer Variables Satisfying Linear Congruences", *SIAM J. Comput.*, Vol. 17, No. 2, pp. 262-280, 1988. Combines early papers from *FOCS84* and *STOC85* (by Frieze, Kannan and Lagarias, and Hastad and Shamir, resp.).

[G87b] Goldreich, O., "Towards a Theory of Software Protection and Simulation by Oblivious RAMs", *19th STOC*, pp. 182-194, 1987.

[K69] Knuth, D.E., *The Art of Computer Programming*, Vol. 2, Addison-Wesley, Reading, Mass., 1969.

[Kr88] Krawczyk, H., "How to Predict Congruential Generators", TR-533, Computer Science Dept., Technion, Haifa, Israel, 1988. To appear in *J. of Algorithms*.

[LR88] J.C. Lagarias, and J. Reeds, "Unique Extrapolation of Polynomial Recurrences", *SIAM J. Comput.*, Vol. 17, No. 2, pp. 342-362, 1988.

[LLS87] Lichtenstein, D., N. Linial, and M. Saks, "Imperfect Random Sources and Discrete Control Processes", *19th STOC*, pp. 169-177, 1987.

[MH78] see category 8.

[OS85] Okamoto, T., and A. Shiraishi, "A Fast Signature Scheme Based on Quadratic Inequalities", *Proc. of 1985 Symp. on Security and Privacy*, April 1985, Oakland, Cal.

[OSS84] Ong, H., C.P. Schnorr, and A. Shamir, "An Efficient Signature Scheme Based on Quadratic Equations", *16th STOC*, pp. 208-216, 1984.

[OSS84b] Ong, H., C.P. Schnorr, and A. Shamir, "Efficient Signature Schemes Based on Polynomial Equations", *Crypto84*, proceedings, Springer-Verlag, Lecture Notes in Computer Science, vol. 196, pp. 37-46, 1985.

[O89] Ostrovsky, R., "An Efficient Software Protection Scheme", in preparations.

[Pol84] Pollard, J.M., "Solution of $x^2 + ky^2 \equiv m (mod\, n)$, with Application to Digital Signatures", preprint, 1984.

[R83] Rabin, M.O., "Randomized Byzantine Agreement", *24th FOCS*, pp. 403-409, 1983.

[SV84] Santha, M., and U.V. Vazirani, "Generating Quasi-Random Sequences from Slightly-Random Sources", *25th Symp. on Foundation of Computer Science*, pp. 434-440, 1984.

[S82] Shamir, A., "A Polynomial-Time Algorithm for Breaking the Merkle-Hellman Cryptosystem", *23rd FOCS*, pp. 145-152, 1982.

[S87] Stern, J., "Secret Linear Congruential Generators are not Cryptographically Secure", *28th FOCS*, pp. 421-426, 1987.

[V85] U.V. Vazirani, "Towards a Strong Communication Complexity Theory or Generating Quasi-Random Sequences from Two Communicating Slightly-Random Sources", *Proc. 17th ACM Symp. on Theory of Computing*, 1985, pp. 366-378.

[V87] U.V. Vazirani, "Efficiency Considerations in Using Semi-random Sources", *Proc. 19th ACM Symp. on Theory of Computing*, 1987, pp. 160-168.

[VV85] U.V. Vazirani, and V.V. Vazirani, "Random Polynomial Time is equal to Slightly-Random Polynomial Time", *26th Symp. on Foundation of Computer Science*, pp. 417-428, 1985.

A.8 Historical Background

An inspection of the references listed above reveals that all these works were initiated in the 80's and began to appear in the literature in 1982 (e.g. [GM82]). However, previous work had tremendous influence on these works of the 80's. The influence took the form of setting intuitive goals, providing basic techniques, and suggesting potential solutions which served as a basis for constructive criticism (leading to robust approaches).

8.1. Classic Cryptography

Answering the fundamental question of classic cryptography in a gloomy way (i.e. it is *impossible* to design a code that cannot be broken), Shannon suggested a modification to the question [S49]. Rather than asking whether it is *possible* to break the code, one should ask whether it is *feasible* to break it. A code should be considered good if it cannot be broken when investing work which is in reasonable proportion to the work required of the legal parties using the code.

8.2. New Directions in Cryptography

Prospects of commercial application were the trigger for the beginning of civil investigations of encryption schemes. The DES designed in the early 70's has adopted the new paradigm: it is clearly *possible* but supposely *infeasible* to break it.

Following the challenge of constructing and analyzing new encryption schemes came new questions like how to exchange keys over an insecure channel [M78]. New concepts were invented: *digital signatures* [R77, DH76], *public-key cryptosystems* and *one-way functions* [DH76]. First implementations of these concepts were suggested in [MH78, RSA78, R79].

Cryptography was explicitly related to complexity theory in [Br79, EY80, Lem79]: it was understood that problems related to breaking a cryptographic scheme cannot be $\mathcal{NP}$-complete and that $\mathcal{NP}$-hardness is a poor evidence for cryptographic security. Techniques as "n-out-of-$2n$ verification" [R77] and secret sharing [S79] were introduced (and indeed were used extensively in subsequent research).

8.3. At the Dawn of a New Era

Early investigations of cryptographic protocols revealed the inadequacy of imprecise notions of security and the subtleties involved in designing cryptographic protocols. In particular, problems as *coin tossing over telephone* [B82a], *exchange of secrets* and *oblivious transfer* were formulated [R81, B82b] (cf. [EGL82]). Doubts concerning the security of "mental poker" protocol of [SRA79] led to the current notion of secure encryption [GM82] and to

concepts as computational indistinguishability. Doubts concerning the Oblivious Transfer protocol of [R81] led to the concept of zero-knowledge [GMR85] (early versions date to March 1982).

An alternative approach to the security of cryptographic protocols was suggested in [DY81] (see also [DEK82]), but it turned out that it is much too difficult to test whether a protocol is secure [EG83]. Fortunately, tools for constructing secure protocols do exist (see [Y86, GMW87])!

references

[B82a] Blum, M., "Coin Flipping by Phone", *IEEE Spring COMPCOM*, pp. 133-137, February 1982. See also *SIGACT News*, Vol. 15, No. 1, 1983.

[B82b] Blum, M., "How to Exchange Secret Keys", Memo. No. UCB/ERL M81/90. *ACM Trans. Comput. Sys.*, Vol. 1, pp. 175-193, 1983.

[Br79] Brassard, G., "A Note on the Complexity of Cryptography", *IEEE Trans. on Inform. Th.*, Vol. 25, pp. 232-233, 1979.

[DH76] W. Diffie, and M. E. Hellman, "New Directions in Cryptography", *IEEE transactions on Info. Theory*, IT-22 (Nov. 1976), pp. 644-654

[DEK82] Dolev, D., S. Even, and R. Karp, "On the Security of Ping-Pong Protocols", *Advances in Cryptology: Proceedings of Crypto82*, Plenum Press, pp. 177-186, 1983.

[DY81] Dolev, D., and A.C. Yao, "On the Security of Public-Key Protocols", *IEEE Trans. on Inform. Theory*, Vol. 30, No. 2, pp. 198-208, 1983. Early version in *FOCS81*.

[EGL82] Even, S., O. Goldreich, and A. Lempel, "A Randomized Protocol for Signing Contracts", *CACM*, Vol. 28, No. 6, 1985, pp. 637-647. Extended abstract in *Crypto82*.

[EG83] Even, S., and O. Goldreich, "On the Security of Multi-party Ping-Pong Protocols", *24th FOCS*, pp. 34-39, 1983.

[EY80] Even, S., and Y. Yacobi, "Cryptography and NP-Completeness", *7th ICALP* proceedings, Lecture Notes in Computer Science, Vol. 85, Springer Verlag, pp. 195-207, 1980. See also later version by Even, Selman, and Yacobi (titled: "The Complexity of Promise Problems with Applications to Public-Key Cryptography") in *Inform. and Control*, Vol. 61, pp. 159-173, 1984.

[GMW87] see main references.

[GM82] see main references.

[GMR85] see main references.

[Lem79] Lempel, A., "Cryptography in Transition", *Computing Surveys*, Dec. 1979.

[M78] Merkle, R.C., "Secure Communication over Insecure Channels", *CACM*, Vol. 21, No. 4, pp. 294-299, 1978.

[MH78] Merkle, R.C., and M.E. Hellman, "Hiding Information and Signatures in Trapdoor Knapsacks", *IEEE Trans. Inform. Theory*, Vol. 24, pp. 525-530, 1978.

[R77] M.O. Rabin, "Digitalized Signatures", *Foundations of Secure Computation*, Academic Press, R.A. DeMillo et. al. eds., 1977.

[R79] M.O. Rabin, "Digitalized Signatures and Public Key Functions as Intractable as Factoring", MIT/LCS/TR-212, 1979.

[R81] Rabin, M.O., "How to Exchange Secrets by Oblivious Transfer", unpublished manuscript, 1981.

[RSA78] R. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public Key Cryptosystems", *Comm. ACM*, Vol. 21, Feb. 1978, pp 120-126

[S79] Shamir, A., "How to Share a Secret", *CACM*, Vol. 22, 1979, pp. 612-613.

[S83] A. Shamir, "On the Generation of Cryptographically Strong Pseudorandom Sequences", *ACM Transaction on Computer Systems*, Vol. 1, No. 1, February 1983, pp. 38-44.

[SRA79] Shamir, A., R.L. Rivest, and L. Adleman, "Mental Poker", MIT/LCS report TM-125, 1979.

[S49] Shannon, C.E., "Communication Theory of Secrecy Systems", *Bell Sys. Tech. J.*, 28, pp. 656-715, 1949.

[Y86] see category 6.