Some Facets of Complexity Theory and Cryptography: A Five-Lectures Tutorial

JÖRG ROTHE
Heinrich-Heine-Universität Düsseldorf

In this tutorial, selected topics of cryptology and of computational complexity theory are presented. We give a brief overview of the history and the foundations of classical cryptography, and then move on to modern public-key cryptography. Particular attention is paid to cryptographic protocols and the problem of constructing key components of protocols such as one-way functions. A function is one-way if it is easy to compute, but hard to invert. We discuss the notion of one-way functions both in a cryptographic and in a complexity-theoretic setting. We also consider interactive proof systems and present some interesting zero-knowledge protocols. In a zero-knowledge protocol one party can convince the other party of knowing some secret information without disclosing any bit of this information. Motivated by these protocols, we survey some complexity-theoretic results on interactive proof systems and related complexity classes.

Categories and Subject Descriptors: E.3 [**Data Encryption**]: Public-key Cryptosystems; F.1.3 [**Computation by Abstract Devices**]: Complexity Measures and Classes—*Complexity hierarchies*; *Relations among complexity classes*; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Computations on discrete structures*

General Terms: Theory, Security, Algorithms
Additional Key Words and Phrases: complexity theory, public-key cryptography, secret-key agreement, digital signatures, interactive proof systems, zero-knowledge protocols, one-way functions

Author's address: J. Rothe, Institut für Informatik, Heinrich-Heine-Universität Düsseldorf, 40225 Düsseldorf, Germany. Email address: rothe@cs.uni-duesseldorf.de.

Contents

Outline of the Tutorial

This tutorial consists of five lectures on cryptography, based on the lecture notes for a course on this subject given by the author in August, 2001, at the 11th Jyväskylä Summer School in Jyväskylä, Finland. As the title suggests, a particular focus of this tutorial is to emphasize the close relationship between cryptography and complexity theory. The material presented here is not meant to be a comprehensive study or a complete survey of (the intersection of) these fields. Rather, five vivid topics from those fields are chosen for exposition, and from each topic chosen, some gems—some particularly important, central, beautiful results—are presented. Needless to say, the choice of topics and of results selected for exposition is based

on the author's personal tastes and biases.

The first lecture sketches the history and the classical foundations of cryptography, introduces a number of classical, symmetric cryptosystems, and briefly discusses by example the main objectives of the two opposing parts of cryptology: cryptography, which aims at designing secure ways of encryption, versus cryptanalysis, which aims at breaking existing cryptosystems. Then, we introduce the notion of perfect secrecy for cryptosystems, which dates back to Claude Shannon's pioneering work [Sha49] on coding and information theory.

The second lecture presents the public-key cryptosystem RSA, which was invented by Rivest, Shamir, and Adleman [RSA78]. RSA is the first public-key cryptosystem developed in the public sector. To describe RSA, some background from number theory is provided in as short a way as possible but to the extent necessary to understand the underlying mathematics. In contrast to the information-theoretical approach of perfect secrecy, the security of RSA is based on the assumption that certain problems from number theory are computationally intractable. Potential attacks on the RSA cryptosystem as well as appropriate countermeasures against them are discussed.

The third lecture introduces a number of cryptographic protocols, including the secret-key agreement protocols of Diffie and Hellman [DH76] and of Rivest and Sherman (see [RS93,RS97]), ElGamal's public-key cryptosystem [ElG85], Shamir's no-key protocol, and the digital signature schemes of Rivest, Shamir, and Adleman [RSA78], ElGamal [ElG85], and Rabi and Sherman [RS93,RS97], respectively. Again, the underlying mathematics and, relatedly, security issues of these protocols are briefly discussed.

A remark is in order here. The protocols presented here are among the most central and important cryptographic protocols, with perhaps two exceptions: the Rivest–Sherman and the Rabi–Sherman protocols. While the secret-key agreement protocol of Diffie and Hellman [DH76] is widely used in practice, that of Rivest and Sherman (see [RS93,RS97]) is not (yet) used in applications and, thus, might appear somewhat exotic at first glance. An analogous comment applies to the Rabi–Sherman digital signature protocol. However, from our point of view, there is some hope that this fact, though currently true, might change in the near future. In Section 3.5, we will discuss the state of the art on the Diffie–Hellman protocol and the Rivest–Sherman protocol, and we will argue that recent progress of results in complexity theory may lead to a significant increase in the cryptographic security and the applicability of the Rivest–Sherman protocol. One line of complexity-theoretic research that is relevant here is presented in Section 5; another line of research is Ajtai's breakthrough result [Ajt96] on the complexity of the shortest lattice vector problem (SVP, for short), which is informally stated in Section 3.5.

The fourth lecture introduces interactive proof systems and zero-knowledge protocols. This area has rapidly developed and flourished in complexity theory and has yielded a number of powerful results. For example, Shamir's famous result [Sha92] characterizes the power of interactive proof systems in terms of classical complexity classes: Interactive proof systems precisely capture the class of problems solvable in polynomial space. Also, the study of interactive proof systems is related to probabilistically checkable proofs, which has yielded novel nonapproximability results for

hard optimization problems; see the survey [Gol97]. Other results about interactive proof systems and the related zero-knowledge protocols have direct applications in cryptography. In particular, zero-knowledge protocols enable one party to convince another party of knowledge of some secret information without conveying any bit of this information. Thus, they are ideal technical tools for authentication purposes. We present two of the classic zero-knowledge protocols: the Goldreich-Micali-Wigderson protocol for graph isomorphism [GMW86,GMW91] and the Fiat-Shamir protocol [FS86] that is based on a number-theoretical problem. For an in-depth treatment of zero-knowledge protocols and many more technical details, the reader is referred to Chapter 4 of Goldreich's book [Gol01b].

The fifth lecture gives an overview on the progress of results that was recently obtained by Hemaspaandra, Pasanen, and this author [HR99,HPR01]. Their work, which is motivated by the Rivest–Sherman and the Rabi–Sherman protocols, studies properties of functions that are used in building these two cryptographic protocols. It is results about these functions that may be useful in quantifying the security of these protocols. In particular, the key building block of the Rivest–Sherman protocol is a strongly noninvertible, associative one-way function. Section 5 presents the result [HR99] on how to construct such a function from the assumption that $P \neq NP$. In addition, recent results on strong noninvertibility are surveyed, including the perhaps somewhat surprising result that if $P \neq NP$ then there exist strongly noninvertible functions that in fact are invertible [HPR01]. These results are obtained in the *worst-case* complexity model, which is relevant and interesting in a complexity-theoretic setting, but useless in applied cryptography. For cryptographic applications, one would need to construct such functions based on the *average-case* complexity model, under plausible assumptions. Hence, the most challenging open research question related to strongly noninvertible, associative one-way functions is to find some evidence that they exist even in the average-case model. As noted above, our hope of obtaining such a result is based on recent progress on the shortest lattice vector problem accomplished by Ajtai [Ajt96]. Roughly speaking, Ajtai proved that this problem is as hard in the average-case as it is in the worst-case model. Based on this result, Ajtai and Dwork [AD97] designed a public-key cryptosystem whose security is based merely on worst-case assumptions. Ajtai's breakthrough results, his techniques, and their cryptographic applications are not covered in this tutorial. We refer to the nice surveys by Cai [Cai99] and, more recently, by Kumar and Sivakumar [KS01] and Nguyen and Stern [NS01] on the complexity of SVP and the use of lattices in crytography.

The tutorial is suitable for graduate students with some background in computer science and mathematics and may also be accessible to interested undergraduate students. Since it is organized in five essentially independent, self-contained lectures, it is also possible to present only a proper subset of these lectures. The only dependencies occurring between lectures are that some of the number-theoretical background given in Section 2 is also used in Section 3, and that the Rivest–Sherman secret-key agreement protocol and the Rabi–Sherman digital signature protocol presented in Section 3 motivate the investigations in Section 5. This last section contains perhaps the technically most challenging material, which in part is presented on an expert level with the intention of guiding the reader towards an

active field of current research.

There are a number of textbooks and monographs on cryptography that cover various parts of the field in varying depth, such as the books by Goldreich [Gol99,Gol01b], Salomaa [Sal96], Stinson [Sti95], and Welsh [Wel98]. Schneier's book [Sch96] provides a very comprehensive collection of literally all notions and concepts known in cryptography, which naturally means that the single notions and concepts cannot be treated in mathematical detail there, but the interested reader is referred to an extraordinarily large bibliography for such an in-depth treatment. Singh [Sin99] wrote a very charming, easy-to-read, interesting book about the history of cryptography from its ancient roots to its modern and even futuristic branches such as quantum cryptography. An older but still valuable source is Kahn's book [Kah67]. We conclude this list, without claiming it to be complete, with the books by Bauer [Bau00], Beutelspacher et al. [BSW01,Beu94], and Buchmann [Buc01].

1. CRYPTOSYSTEMS AND PERFECT SECRECY

1.1 Classical Cryptosystems

The notion of a cryptosystem is formally defined as follows.

Definition 1.1 *Cryptosystem.*

—A *cryptosystem* is a quintuple $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ such that:

(1) $\mathcal{P}$, $\mathcal{C}$, and $\mathcal{K}$ are finite sets, where

     $\mathcal{P}$ is the *plain text space* or *clear text space*;

     $\mathcal{C}$ is the *cipher text space*;

     $\mathcal{K}$ is the *key space*.

Elements of $\mathcal{P}$ are referred to as plain text (or clear text), and elements of $\mathcal{C}$ are referred to as cipher text. A *message* is a string of plain text symbols.

(2) $\mathcal{E} = \{E_k \mid k \in \mathcal{K}\}$ is a family of functions $E_k : \mathcal{P} \to \mathcal{C}$ that are used for encryption, and $\mathcal{D} = \{D_k \mid k \in \mathcal{K}\}$ is a family of functions $D_k : \mathcal{C} \to \mathcal{P}$ that are used for decryption.

(3) For each key $e \in \mathcal{K}$, there exists a key $d \in \mathcal{K}$ such that for each $p \in \mathcal{P}$:

$$D_d(E_e(p)) = p. \tag{1.1}$$

—A *cryptosystem* is called *symmetric* (or *"private-key"*) if $d = e$, or if d can at least be "easily" computed from e.

—A *cryptosystem* is called *asymmetric* (or *"public-key"*) if $d \neq e$, and it is "computationally infeasible in practice" to compute d from e. Here, d is the *private key*, and e is the *public key*.

At times, different key spaces are used for encryption and for decryption, which results in a slight modification of the above definition.

We now present and discuss some examples of classical cryptosystems. Consider the English alphabet $\Sigma = \{A, B, \ldots, Z\}$. To carry out the arithmetic modulo 26 with letters as if they were numbers, we identify Σ with $\mathbb{Z}_{26} = \{0, 1, \ldots, 25\}$; thus, 0 represents A and 1 represents B, and so on. This encoding of the plain text alphabet by integers and the decoding of $\mathbb{Z}_{26}$ back to Σ is not part of the actual

encryption and decryption, respectively. It will be used for the next three examples. Note that messages are elements of Σ^*, where Σ^* denotes the set of strings over Σ.

Example 1.2 *Caesar cipher, a monoalphabetic symmetric cryptosystem.*

Let $\mathcal{K} = \mathbb{Z}_{26}$, and let $\mathcal{P} = \mathcal{C} = \Sigma$. The *Caesar cipher* encrypts messages by shifting (modulo 26) each character of the plain text by the same number k of letters in the alphabet, where k is the key. Shifting each character of the cipher text back using the same key k reveals the original message:

—For each $e \in \mathbb{Z}_{26}$, define the encryption function $E_e : \Sigma \to \Sigma$ by

$$E_e(p) = (p + e) \mod 26,$$

where addition with e modulo 26 is carried out character-wise, i.e., each character $m_i \in \Sigma$ of a message $m \in \Sigma^*$ is shifted by e positions to $m_i + e \mod 26$. For example, using the key $e = 11 = L$, the message "SUMMER" will be encrypted as "DFXXPC."

—For each $d \in \mathbb{Z}_{26}$, define the decryption function $D_d : \Sigma \to \Sigma$ by

$$D_d(c) = (c - d) \mod 26,$$

where subtraction by e modulo 26 again is carried out character-wise. Hence, $d = e$. For example, decrypting the cipher text "DNSZZW" with the key $d = 11$ reveals the plain text "SCHOOL."

Since the key space is very small, breaking the Caesar cipher is very easy. It is vulnerable even to *"cipher-text-only attacks,"* i.e., an attacker given enough cipher text c can easily check the 26 possible keys to see which one yields a meaningful plain text. Note that the given cipher text should contain enough letters to enable a unique decryption.

The Caesar cipher is a monoalphabetic cryptosystem, since it replaces each given plain text letter, wherever in the message it occurs, by the same letter of the cipher text alphabet. In contrast, the French cryptographer and diplomat Blaise de Vigenère (1523–1596) proposed a polyalphabetic cryptosystem, which is much harder to break. Vigenère's system builds on earlier work by the Italian mathematician Leon Battista Alberti (born in 1404), the German abbot Johannes Trithemius (born in 1492), and the Italian scientist Giovanni Porta (born in 1535), see [Sin99]. It works like the Caesar cipher, except that the cipher text letter encrypting any given plain text letter X varies with the position of X in the plain text.

More precisely, one uses for encryption and decryption a *Vigenère square*, which consists of 26 rows with 26 columns each. Every row contains the 26 letters of the alphabet, shifted by one from row to row, i.e., the rows and columns may be viewed as a Caesar encryption of the English alphabet with keys 0, 1, …, 25. Given a message $m \in \Sigma^*$, one first chooses a key $k \in \Sigma^*$, which is written above the message m, symbol by symbol, possibly repeating k if k is shorter than m until every character of m has a symbol above it. Denoting the ith letter of any string w by w_i, each letter m_i of m is then encrypted as in the Caesar cipher, using the row of the Vigenère square that starts with k_i, where k_i is the key letter right above m_i. Below, we describe the Vigenère system formally and give an example of a concrete encryption.

Example 1.3 *Vigenère cipher, a polyalphabetic symmetric cryptosystem.*

For fixed $n \in \mathbb{N}$, let $\mathcal{K} = \mathcal{P} = \mathcal{C} = \mathbb{Z}_{26}^n$. Messages $m \in \Sigma^*$, where Σ again is the English alphabet, are split into blocks of length n and are encrypted block-wise. The *Vigenère cipher* is defined as follows.

—For each $e \in \mathbb{Z}_{26}^n$, define the encryption function $E_e : \mathbb{Z}_{26}^n \to \mathbb{Z}_{26}^n$ by

$$E_e(p) = (p + e) \mod 26,$$

where addition with e modulo 26 is carried out character-wise, i.e., each character $p_i \in \Sigma$ of a plain text $p \in \mathcal{P}$ is shifted by e_i positions to $p_i + e_i \mod 26$.

—For each $d \in \mathbb{Z}_{26}^n$, define the decryption function $D_d : \mathbb{Z}_{26}^n \to \mathbb{Z}_{26}^n$ by

$$D_d(c) = (c - d) \mod 26,$$

where subtraction modulo 26 again is carried out character-wise. As in the Caesar cipher, $d = e$.

For example, choose the word $k = $ ENGLISH to be the key. Suppose we want to encrypt the message $m = $ FINNISHISALLGREEKTOGERMANS,[1] omitting the spaces between words. Table I shows how each plain text letter is encrypted, yielding the cipher text c. For instance, the first letter of the message, "F," corresponds to the first letter of the key, "E." Hence, the intersection of the "F"-column with the "E"-row of the Vigenère square gives the first letter, "J," of the cipher text.

k	E	N	G	L	I	S	H	E	N	G	L	I	S	H	E	N	G	L	I	S	H	E	N	G	L	I
m	F	I	N	N	I	S	H	I	S	A	L	L	G	R	E	E	K	T	O	G	E	R	M	A	N	S
c	J	V	T	Y	Q	K	O	M	F	G	W	T	Y	Y	I	R	Q	E	W	Y	L	V	Z	G	Y	A

Table I.   An example of encryption by the Vigenère cipher.

Our last example of a classical, historically important cryptosystem is the Hill cipher, which was invented by Lester Hill in 1929. It is based on linear algebra and, like the Vigenère cipher, is an affine linear block cipher.

Example 1.4 *Hill cipher, a symmetric cryptosystem and a linear block cipher.*

For fixed $n \in \mathbb{N}$, the key space $\mathcal{K}$ is the set of all invertible $n \times n$ matrices in $\mathbb{Z}_{26}^{n \times n}$. Again, $\mathcal{P} = \mathcal{C} = \mathbb{Z}_{26}^n$ and messages $m \in \Sigma^*$ are split into blocks of length n and are encrypted block-wise. All arithmetic operations are carried out modulo 26. The *Hill cipher* is defined as follows.

—For each $K \in \mathcal{K}$, define the encryption function $E_K : \mathbb{Z}_{26}^n \to \mathbb{Z}_{26}^n$ by

$$E_K(p) = K \cdot p \mod 26,$$

where $\cdot$ denotes matrix multiplication modulo 26.

---

[1] From this example we not only learn how the Vigenère cipher works, but also that using a language such as Finnish, which is not widely used, often makes illegal decryption harder, and thus results in a higher level of security. This is not a purely theoretical observation. During World War II, the US Navy transmitted important messages using the language of the Navajos, a Native American tribe. The "Navajo Code" was never broken by the Japanese code-breakers, see [Sin99].

—Letting K^{-1} denote the inverse matrix of K, the decryption function $D_{K^{-1}} : \mathbb{Z}_{26}^n \to \mathbb{Z}_{26}^n$ is defined by

$$D_{K^{-1}}(c) = K^{-1} \cdot c \mod 26.$$

Since K^{-1} can easily be computed from K, the Hill cipher is a symmetric cryptosystem. It is also the most general linear block cipher.

Concrete examples of messages encrypted by the Hill cipher can be found in, e.g., [Sal96].

Affine linear block ciphers are easy to break by "*known-plain-text attacks*," i.e., for an attacker who knows some sample plain texts with the corresponding encryptions, it is not too hard to find the key used to encrypt these plain texts. They are even more vulnerable to "*chosen-plain-text attacks*," where the attacker can choose some pairs of corresponding plain texts and encryptions, which may be useful if there are reasonable conjectures about the key used.

The method of frequency counts is often useful for decrypting messages. It exploits the redundancy of the natural language used for plain text messages. For example, in many languages the letter "E" occurs, statistically significant, most frequently, with a percentage of 12.31% in English, of 15.87% in French, and even of 18.46% in German, see [Sal96]. Some languages have other letters that occur with the highest frequency; for example, "A" is the most frequent letter in average Finnish texts, with a percentage of 12.06% [Sal96].

In 1863, the German cryptanalyst Friedrich Wilhelm Kasiski found a method to break the Vigenère cipher. Singh [Sin99] attributes this achievement also to an unpublished work, done probably around 1854, by the British genius and eccentric Charles Babbage. The books by Salomaa [Sal96] and Singh [Sin99] describe Kasiski's and Babbage's method. It marks a breakthrough in the history of cryptanalysis, because previously the Vigenère cipher was considered unbreakable. In particular, like similar periodic cryptosystems with an unknown period, the Vigenère cipher appeared to resist cryptanalysis by counting and analysing the frequency of letters in the cipher text. Kasiski showed how to determine the period from repetitions of the same substring in the cipher text.

In light of Kasiski's and Babbage's achievement, it is natural to ask whether there exist any cryptosystems that guarantee *perfect secrecy*. We turn to this question in the next section that describes some of the pioneering work of Claude Shannon [Sha49], who laid the foundations of modern coding and information theory.

1.2   Conditional Probability and Bayes's Theorem

To discuss perfect secrecy of cryptosystems in mathematical terms, we first need some preliminaries from elementary probability theory.

Definition 1.5. Let A and B be events with $\Pr(B) > 0$.

—The *probability that A occurs under the condition that B occurs* is defined by

$$\Pr(A \mid B) = \frac{\Pr(A \cap B)}{\Pr(B)}.$$

—A and B are *independent* if $\Pr(A \cap B) = \Pr(A) \Pr(B)$ (equivalently, if $\Pr(A \mid B) = \Pr(A)$).

LEMMA 1.6 BAYES'S THEOREM. *Let A and B be events with $\Pr(A) > 0$ and $\Pr(B) > 0$. Then,*

$$\Pr(B)\Pr(A \mid B) = \Pr(A)\Pr(B \mid A).$$
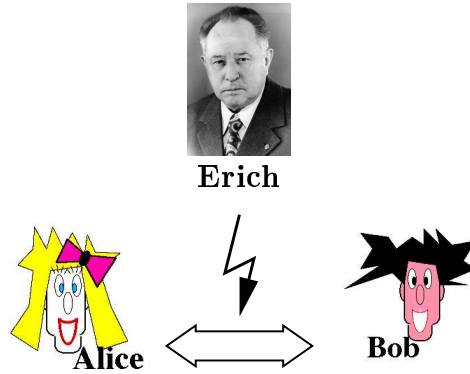
PROOF. By definition,

$$\Pr(B)\Pr(A \mid B) = \Pr(A \cap B) = \Pr(B \cap A) = \Pr(A)\Pr(B \mid A).$$

∎

1.3 Perfect Secrecy: Shannon's Theorem

Consider the following scenario:

Erich

Alice ⟺ **Bob**

Using a cryptosystem $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$, Alice and Bob are communicating over an insecure channel in the presence of eavesdropper Erich. Recall that $\mathcal{P}$, $\mathcal{C}$, and $\mathcal{K}$ are finite sets. Erich reads a cipher text, $c \in \mathcal{C}$, and tries to get some information about the corresponding plain text, $p \in \mathcal{P}$. The plain texts are distributed on $\mathcal{P}$ according to a probability distribution $\Pr_{\mathcal{P}}$ that may depend on the language used. For each new plain text, Alice chooses a new key from $\mathcal{K}$ that is independent of the plain text to be encrypted. The keys are distributed according to a probability distribution $\Pr_{\mathcal{K}}$ on $\mathcal{K}$. The distributions $\Pr_{\mathcal{P}}$ and $\Pr_{\mathcal{K}}$ induce a probability distribution $\Pr = \Pr_{\mathcal{P} \times \mathcal{K}}$ on $\mathcal{P} \times \mathcal{K}$. Thus, for each plain text p and each key k,

$$\Pr(p, k) = \Pr_{\mathcal{P}}(p)\Pr_{\mathcal{K}}(k)$$

is the probability that the plain text p is encrypted with the key k, where p and k are independent.

$\Pr(p) = \Pr_{\mathcal{P}}(p)$ is the probability that the plain text p will be encrypted. Similarly, $\Pr(k) = \Pr_{\mathcal{K}}(k)$ is the probability that the key k will be used. Let c be another random variable whose distribution is determined by the system used. Then, $\Pr(p \mid c)$ is the probability that p is encrypted under the condition that c is received. Erich knows the cipher text c, and he knows the probability distribution $\Pr_{\mathcal{P}}$, since he knows the language used by Alice and Bob.

Definition 1.7. A cryptosystem $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ provides *perfect secrecy* if and only if

$$(\forall p \in \mathcal{P})\,(\forall c \in \mathcal{C})\,[\Pr(p \mid c) = \Pr(p)].$$

That is, a cryptosystem achieves perfect secrecy if the event that some plain text p is encrypted and the event that some cipher text c is received are independent: Erich learns nothing about p from knowing c. The following example of a cryptosystem that does not provide perfect secrecy is due to Buchmann [Buc01].

Example 1.8 *Perfect secrecy.* Let $\mathcal{P}$, $\mathcal{C}$, and $\mathcal{K}$ be given such that:

—$\mathcal{P} = \{0, 1\}$, where $\Pr(0) = \frac{1}{4}$ and $\Pr(1) = \frac{3}{4}$;

—$\mathcal{K} = \{A, B\}$, where $\Pr(A) = \frac{1}{4}$ and $\Pr(B) = \frac{3}{4}$;

—$\mathcal{C} = \{a, b\}$.

It follows that, for example, the probability that a "1" occurs and is encrypted with the key B is:

$$\Pr(1, B) = \Pr(1) \cdot \Pr(B) = \frac{3}{4} \cdot \frac{3}{4} = \frac{9}{16}.$$

Let the encryption functions be given by:

$$E_A(0) = a; \ \ E_A(1) = b; \ \ E_B(0) = b; \ \ E_B(1) = a.$$

Hence, the probability that the cipher text a occurs is:

$$\Pr(a) = \Pr(0, A) + \Pr(1, B) = \frac{1}{16} + \frac{9}{16} = \frac{5}{8}.$$

Similarly, the probability that the cipher text b occurs is:

$$\Pr(b) = \Pr(1, A) + \Pr(0, B) = \frac{3}{16} + \frac{3}{16} = \frac{3}{8}.$$

Then, for each pair $(p, c) \in \mathcal{P} \times \mathcal{C}$, the conditional probability $\Pr(p \,|\, c)$ is:

$$\Pr(0 \,|\, a) = \frac{\Pr(0, A)}{\Pr(a)} = \frac{\frac{1}{16}}{\frac{5}{8}} = \frac{1}{10}; \qquad \Pr(0 \,|\, b) = \frac{\Pr(0, B)}{\Pr(b)} = \frac{\frac{3}{16}}{\frac{3}{8}} = \frac{1}{2};$$

$$\Pr(1 \,|\, a) = \frac{\Pr(1, B)}{\Pr(a)} = \frac{\frac{9}{16}}{\frac{5}{8}} = \frac{9}{10}; \qquad \Pr(1 \,|\, b) = \frac{\Pr(1, A)}{\Pr(b)} = \frac{\frac{3}{16}}{\frac{3}{8}} = \frac{1}{2}.$$

In particular, it follows that

$$\Pr(0) = \frac{1}{4} \neq \frac{1}{10} = \Pr(0 \,|\, a),$$

and thus the given cryptosystem does not provide perfect secrecy: If Erich sees the cipher text a, he can be pretty sure that the encrypted plain text was a "1."

THEOREM 1.9 SHANNON [SHA49]. *Let* $S = (\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ *be a cryptosystem with* $||\mathcal{C}|| = ||\mathcal{K}||$ *and* $\Pr(p) > 0$ *for each* $p \in \mathcal{P}$. *Then*, S *provides perfect secrecy if and only if*

(1) $\Pr_{\mathcal{K}}$ *is the uniform distribution, and*

(2) *for each* $p \in \mathcal{P}$ *and for each* $c \in \mathcal{C}$, *there exists a unique key* $k \in \mathcal{K}$ *with* $E_k(p) = c$.

PROOF. Assume that S provides perfect secrecy. We show that the conditions (1) and (2) hold.

Condition (2): Fix a plain text $p \in \mathcal{P}$. Suppose that there is a cipher text $c \in \mathcal{C}$ such that for all $k \in \mathcal{K}$, it holds that $E_k(p) \neq c$. Thus,

$$\Pr(p) \neq 0 = \Pr(p \,|\, c),$$

which implies that S does not provide perfect secrecy, a contradiction. Hence,

$$(\forall c \in \mathcal{C})\,(\exists k \in \mathcal{K})\,[E_k(p) = c].$$

Now, $||\mathcal{C}|| = ||\mathcal{K}||$ implies that each cipher text $c \in \mathcal{C}$ has a unique key k with $E_k(p) = c$.

Condition (1): Fix a cipher text $c \in \mathcal{C}$. For $p \in \mathcal{P}$, let $k(p)$ be the unique key k with $E_k(p) = c$. By Bayes's theorem, for each $p \in \mathcal{P}$, we have:

$$\Pr(p \,|\, c) = \frac{\Pr(c \,|\, p)\,\Pr(p)}{\Pr(c)} = \frac{\Pr(k(p))\,\Pr(p)}{\Pr(c)}. \tag{1.2}$$

Since S provides perfect secrecy, we have $\Pr(p \,|\, c) = \Pr(p)$. By Equation (1.2), this implies $\Pr(k(p)) = \Pr(c)$, and this equality holds independently of p.

Hence, the probabilities $\Pr(k)$ are equal for all $k \in \mathcal{K}$, which implies $\Pr(k) = \frac{1}{||\mathcal{K}||}$. Thus, $\Pr_{\mathcal{K}}$ is the uniform distribution.

Conversely, suppose that conditions (1) and (2) hold. We show that S provides perfect secrecy. Let $k = k(p, c)$ be the unique key k with $E_k(p) = c$. By Bayes's theorem, it follows that

$$
\begin{aligned}
\Pr(p \,|\, c) &= \frac{\Pr(p)\,\Pr(c \,|\, p)}{\Pr(c)} \\
&= \frac{\Pr(p)\,\Pr(k(p, c))}{\sum_{q \in \mathcal{P}} \Pr(q)\,\Pr(k(q, c))}.
\end{aligned}
\tag{1.3}
$$

Since all keys are uniformly distributed, it follows that

$$\Pr(k(p, c)) = \frac{1}{||\mathcal{K}||}.$$

Moreover, we have that

$$\sum_{q \in \mathcal{P}} \Pr(q)\,\Pr(k(q, c)) = \frac{\sum_{q \in \mathcal{P}} \Pr(q)}{||\mathcal{K}||} = \frac{1}{||\mathcal{K}||}.$$

Substituting this equality in Equation (1.3) gives:

$$\Pr(p \,|\, c) = \Pr(p).$$

Hence, S provides perfect secrecy. ∎

1.4 Vernam's One-Time Pad

The Vernam one-time pad is a symmetric cryptosystem that does provide perfect secrecy. It was invented by Gilbert Vernam in 1917,[2] and is defined as follows. Let

---

[2]Slightly differing from the system described here, Vernam's actual invention was a system with a finite period and hence did not provide perfect secrecy; see Kahn [Kah67] on this point.

$\mathcal{P} = \mathcal{C} = \mathcal{K} = \{0,1\}^n$ for some $n \in \mathbb{N}$. For $k \in \{0,1\}^n$, define

—the encryption function $E_k : \{0,1\}^n \rightarrow \{0,1\}^n$ by

$$E_k(p) = p \oplus k \mod 2 \text{ , and}$$

—the decryption function $D_k : \{0,1\}^n \rightarrow \{0,1\}^n$ by

$$D_k(c) = c \oplus k \mod 2,$$

where $\oplus$ denotes bit-wise addition modulo 2. The keys are uniformly distributed on $\{0,1\}^n$. Note that for each plain text p a new key k is chosen from $\{0,1\}^n$.

By Shannon's Theorem, the one-time pad provides perfect secrecy, since for each plain text $p \in \mathcal{P}$ and for each cipher text $c \in \mathcal{C}$, there exists a unique key $k \in \mathcal{K}$ with $c = p \oplus k$, namely the string $k = c \oplus p$.

However, the one-time pad has major disadvantages that make it impractical to use in most concrete scenarios: To obtain perfect secrecy, every key can be used only once, and it must be at least as long as the plain text to be transmitted. Surely, since for every communication a new secret key at least as long as the plain text must be transmitted, this results in a vicious circle. Despite these drawbacks, for the perfect secrecy it provides, the one-time pad has been used in real-world applications such as, allegedly, the hotline between Moscow and Washington, see [Sim79, p. 316].

2. RSA CRYPTOSYSTEM

The RSA cryptosystem, named after its inventors Ron Rivest, Adi Shamir, and Leonard Adleman, is the first public-key cryptosystem [RSA78]. It is still widely used in cryptographic applications today. Again, the scenario is that Alice and Bob want to exchange messages over an insecure channel on which Erich is an eavesdropper:

Erich

Alice   **Bob**
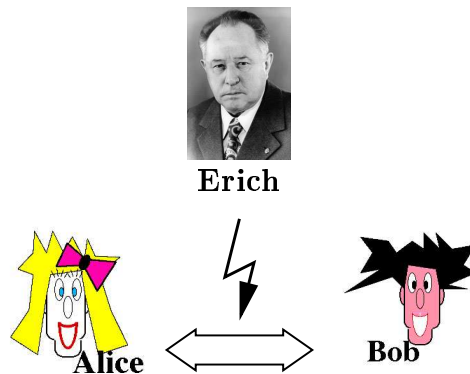
In order to describe how the RSA cryptosystem works, we first need some preliminaries from elementary number theory.

2.1 Euler and Fermat's Theorems

The *greatest common divisor* of two integers a and b is denoted by $\gcd(a,b)$. For $n \in \mathbb{N}$, define the set

$$\mathbb{Z}_n^* = \{i \mid 1 \leq i \leq n-1 \text{ and } \gcd(i,n) = 1\}.$$

The *Euler function* ϕ is defined by $\phi(n) = ||\mathbb{Z}_n^*||$. Note that $\mathbb{Z}_n^*$ is a group (with respect to multiplication) of order $\phi(n)$. The following useful properties of ϕ follow from the definition:

—$\phi(m \cdot n) = \phi(m) \cdot \phi(n)$ for all $m, n \in \mathbb{N}$ with $\gcd(m, n) = 1$, and

—$\phi(p) = p - 1$ for all primes p.

We will specifically use that $\phi(n) = (p - 1)(q - 1)$, where p and q are primes and $n = pq$.

Euler's Theorem below is a special case (for the group $\mathbb{Z}_n^*$) of Langrange's Theorem, which states that for each element g of a finite multiplicative group G having order $|G|$ and the neutral element 1, it holds that $g^{|G|} = 1$.

THEOREM 2.1 EULER. *For each $a \in \mathbb{Z}_n^*$, $a^{\phi(n)} \equiv 1 \mod n$.*

The special case of Euler's Theorem with n being a prime not dividing a is known as Fermat's Little Theorem.

THEOREM 2.2 FERMAT'S LITTLE THEOREM. *If p is a prime and $a \in \mathbb{Z}_p^*$, then $a^{p-1} \equiv 1 \mod p$.*

2.2 RSA

(1) Key generation:

(1) Bob chooses randomly two large primes p and q with $p \neq q$, and computes their product $n = pq$.

(2) Bob chooses a number $e \in \mathbb{N}$ with

$$1 < e < \phi(n) = (p - 1)(q - 1) \quad \text{and} \quad \gcd(e, \phi(n)) = 1. \tag{2.4}$$

(3) Bob computes the unique number d satisfying

$$1 < d < \phi(n) \quad \text{and} \quad e \cdot d \equiv 1 \mod \phi(n). \tag{2.5}$$

That is, d is the inverse of e modulo $\phi(n)$.

(4) The pair (n, e) is Bob's *public key*, and d is Bob's *private key*.

In order to generate two large primes (e.g., primes with 80 digits each) efficiently, one can choose large numbers at random and test them for primality. Since by the Prime Number Theorem, the number of primes not exceeding N is approximately $\frac{N}{\ln N}$, the odds of hitting a prime are good after a reasonably small number of trials. To verify the primality of the number picked, one usually makes use of a randomized polynomial-time primality test such as the Monte Carlo[3] algorithm of Rabin [Rab80] that is related to a deterministic algorithm due to Miller [Mil76]; their primality test is known as the Miller-Rabin test. An alternative, though less popular Monte Carlo algorithm was proposed by Solovay and Strassen [SS77]. The

---

[3]A Monte Carlo algorithm is a randomized algorithm whose "yes" answers are reliable, while its "no" answers may be erroneous with a certain error probability, or vice versa. The corresponding complexity classes are called R and coR, respectively, see [Gil77]. In contrast, a Las Vegas algorithm may for certain sequences of coin flips halt without giving an answer at all, but whenever it gives an answer, this answer is correct. The corresponding class, ZPP = R ∩ coR, was also defined by Gill [Gil77].

Euclid's Algorithm (extended)
  Input: Two integers, b_0 and b_1.
  begin $x_0 := 1$; $y_0 := 0$; $x_1 := 0$; $y_1 := 1$; $i := 1$;
    while b_i does not divide b_{i-1} **do**
      begin
        $q_i := \left\lfloor \frac{b_{i-1}}{b_i} \right\rfloor$;
        $b_{i+1} := b_{i-1} - q_i \cdot b_i$;
        $x_{i+1} := x_{i-1} - q_i \cdot x_i$;
        $y_{i+1} := y_{i-1} - q_i \cdot y_i$;
        $i := i + 1$
      end
    begin output
      $b := b_i$; $(* \; b = \gcd(b_0, b_1) = 1 \; *)$
      $x := x_i$;
      $y := y_i$ $(* \; y$ is the inverse of $b_1 \mod b_0 \; *)$
    end output
  end

Fig. 1.   The extended algorithm of Euclid.

reason why the Solovay-Strassen test is less popular than the Miller-Rabin test is that it is less efficient and less accurate. These two primality tests, along with a careful complexity analysis and the required number-theoretical background, can be found in, e.g., the books by Stinson [Sti95] and Salomaa [Sal96]. Additional primality tests are contained in [Gol01b,Buc01].

> *Note Added in Proof*: Quite recently, Agrawal et al. [AKS02] designed a deterministic polynomial-time algorithm for primality. Their breakthrough result is a milestone in complexity theory and solves a longstanding open problem. It is unlikely, though, that this algorithm will have immediate consequences for cryptographic applications, since Agrawal et al. [AKS02] note that their algorithm has a running time of roughly n^{12} and thus is much less efficient than the probabilistic primality tests currently in use.

We now argue that the keys can be computed efficiently. In particular, the inverse d of e modulo $\phi(n)$ can be computed efficiently via the extended algorithm of Euclid; see Figure 1.

LEMMA 2.3. *On input $b_0 = \phi(n)$ and $b_1 = e$, the extended algorithm of Euclid computes in polynomial time integers x and y such that*

$$x \cdot \phi(n) + y \cdot e \equiv 1 \mod \phi(n).$$

Thus, y is the inverse of e modulo $\phi(n)$, and Bob chooses $d \equiv y \mod \phi(n)$ as his private key.

Example 2.4. Bob chooses the primes $p = 11$ and $q = 23$, and computes their product $n = 253$ and $\phi(253) = 10 \cdot 22 = 220$. The smallest possible e satisfying Equation (2.4) is $e = 3$. The extended algorithm of Euclid yields the following

sequence of b_i, x_i, and y_i:

i	b_i	x_i	y_i	q_i
0	220	1	0	–
1	3	0	1	73
2	1	1	−73	–

Since $1 \cdot 220 + (-73) \cdot 3 = 220 - 219 \equiv 1 \mod 220$, the unique value $d = -73 + 220 = 147$ computed by Bob satisfies Equation (2.5) and is the inverse of $e = 3$ modulo 220.

(2) Encryption: We assume that messages over some alphabet Σ are block-wise encoded as positive integers with a fixed block length. Suppose that $m < n$ is the message Alice wants to send to Bob. Alice knows Bob's public key (n, e) and computes the encryption $c = E_{(n,e)}(m)$ of m, where the encryption function is defined by

$$E_{(n,e)}(m) = m^e \mod n.$$

Performed naively, this computation may require a large number of multiplications, depending on the choice of e. To ensure efficient encryption, we will employ a "fast exponentiation" algorithm called "square-and-multiply," see Figure 2 below.

Square-and-Multiply Algorithm

Input: $m, n, e \in \mathbb{N}$, where $m < n$.

Step 1. Let the binary expansion of the exponent e be given by

$$e = \sum_{i=0}^{k} e_i 2^i, \quad \text{where } e_i \in \{0, 1\}.$$

Step 2. Successively compute m^{2^i}, where $0 \leq i \leq k$, using the equality

$$m^{2^{i+1}} = \left(m^{2^i} \right)^2.$$

   It is not necessary to store the intermediate values of m^{2^i}.

Step 3. In the arithmetic modulo n, compute

$$m^e = \prod_{\substack{i=0 \\ e_i=1}}^{k} m^{2^i}. \tag{2.6}$$

Output: m^e.

Fig. 2.   The square-and-multiply algorithm.

Equation (2.6) in Step 3 of Figure 2 is correct, since

$$m^e = m^{\sum_{i=0}^{k} e_i 2^i} = \prod_{i=0}^{k} \left(m^{2^i} \right)^{e_i} = \prod_{\substack{i=0 \\ e_i=1}}^{k} m^{2^i}.$$

Hence, instead of e multiplications, Alice need compute no more than $2 \log e$ multiplications. Thus, the square-and-multiply method speeds up the encryption exponentially.

Example 2.5. Suppose Alice wants to compute $c = 6^{17} \mod 100$. The binary expansion of the exponent is $17 = 1 + 16 = 2^0 + 2^4$.

(1) Alice successively computes:

$$
\begin{aligned}
6^{2^0} &= 6^1 &&= 6; \\
6^{2^1} &= 6^2 &&= 36; \\
6^{2^2} &= 36^2 &&\equiv -4 \mod 100; \\
6^{2^3} &\equiv (-4)^2 \mod 100 &&\equiv 16 \mod 100; \\
6^{2^4} &\equiv 16^2 \mod 100 &&\equiv 56 \mod 100.
\end{aligned}
$$

(2) Alice computes her cipher text

$$
\begin{aligned}
c = 6^{17} \mod 100 &\equiv 6 \cdot 6^{2^4} \mod 100 \\
&\equiv 6 \cdot 56 \mod 100 \\
&\equiv 36 \mod 100.
\end{aligned}
$$

Note that only four squarings and one multiplication are needed for her to compute the cipher text.

(3) Decryption: Let c, $0 \le c < n$, be the cipher text sent to Bob; c is subject to eavesdropping by Erich. Bob decrypts c using his private key d and the following decryption function:

$$ D_d(c) = c^d \mod n. $$

Again, the fast exponentiation algorithm described in Figure 2 ensures that the legal recipient Bob can decrypt the cipher text efficiently. Thus, the RSA protocol is feasible. To prove that it is correct, we show that Equation (1.1) is satisfied.

Step	Alice		Bob
1			chooses large primes p, q at random, computes $n = pq$ and $\phi(n) = (p-1)(q-1)$, his public key (n, e) with e satisfying Eq. (2.4), and his private key d satisfying Eq. (2.5)
2		$\xleftarrow{(n,e)}$	
3	encrypts message m by computing $$c = m^e \mod n$$		
4		$\xrightarrow{c}$	
5			decrypts cipher text c by computing $$m = c^d = (m^e)^d \mod n$$

Fig. 3. The RSA protocol.

Figure 3 summarizes the single steps of the RSA protocol and displays the information communicated by Alice and Bob that is subject to eavesdropping by Erich.

THEOREM 2.6. *Let (n, e) and d be Bob's public and private key in the RSA protocol. Then, for each message m with $0 \leq m < n$,*

$$m = (m^e)^d \mod n.$$

That is, RSA is a public-key cryptosystem.

PROOF. Since $e \cdot d \equiv 1 \mod \phi(n)$ by Equation (2.5), there exists an integer t such that

$$e \cdot d = 1 + t(p - 1)(q - 1),$$

where $n = pq$. It follows that

$$\begin{aligned}
(m^e)^d = m^{e \cdot d} &= m^{1 + t(p-1)(q-1)} \\
&= m \left(m^{t(p-1)(q-1)} \right) \\
&= m \left(m^{p-1} \right)^{t(q-1)}.
\end{aligned}$$

Hence, we have

$$(m^e)^d \equiv m \mod p, \tag{2.7}$$

since if p divides m then both sides of Equation (2.7) are $0 \mod p$, and if p does not divide m (i.e., $\gcd(p, m) = 1$) then by Fermat's Little Theorem, we have

$$m^{p-1} \equiv 1 \mod p.$$

By a symmetric argument, it holds that

$$(m^e)^d \equiv m \mod q.$$

Since p and q are primes with $p \neq q$, it follows from the Chinese Remainder Theorem (see, e.g., [Knu81] or [Sti95]) that

$$(m^e)^d \equiv m \mod n.$$

Since $m < n$, the claim follows. ■

2.3 RSA Digital Signature Protocol

The RSA public-key cryptosystem described in Section 2.2 can be modified so as to yield a digital signature protocol. Figure 4 shows how the RSA digital signature protocol works. A chosen-plain-text attack on the RSA digital signature scheme, and countermeasures to avoid it, are described in Section 2.4.

2.4 Security of RSA and Possible Attacks on RSA

The security of the RSA cryptosystem strongly depends on whether factoring large integers is intractable. It is widely believed that there is no efficient factoring algorithm, since no such algorithm could be designed as yet, despite considerable efforts in the past. However, it is not known whether the problem of factoring large integers is as hard as the problem of cracking the RSA system.

Step	Alice		Bob
1	chooses $n = pq$, her public key (n, e), and her private key d as in the RSA protocol, see Section 2.2		
2	computes her signature $$\mathrm{sig}_A(m) = m^d \mod n$$ for the message m		
3		$m, \mathrm{sig}_A(m) \Longrightarrow$	
4			verifies Alice's signature by checking the congruence $$m \equiv (\mathrm{sig}_A(m))^e \mod n$$

Fig. 4.   The RSA digital signature protocol.

Here is a list of potential attacks on the RSA system. To preclude these direct attacks, some care must be taken in choosing the primes p and q, the modulus n, the exponent e, and the private key d. For further background on the security of the RSA system and on proposed attacks to break it, the reader is referred to [Bon99,Sha95,KR95,Moo92]. For each attack on RSA that has been proposed in the literature to date, some practical countermeasures are known, rules of thumb that prevent the success of those attacks or, at least, that make their likelihood of success negligibly small.

Factoring attacks:. The aim of the attacker Erich is to use the public key (n, e) to recover the private key d by factoring n, i.e., by computing the primes p and q with $n = pq$. Knowing p and q, he can just like Bob compute $\phi(n) = (p-1)(q-1)$ and thus the inverse d of e modulo $\phi(n)$, using the extended algorithm of Euclid; see Figure 1 and Lemma 2.3. There are various ways in which Erich might mount this type of attack on RSA.

—*Brute-force attack*: Erich might try to factor the modulus n simply by exhaustive search of the complete key space. Choosing n sufficiently large will prevent this type of attack. Currently, it is recommended to use moduli n with at least 768 bits, i.e., the size of 512 bits formerly in use no longer provides adequate protection today. Of course, the time complexity of modular exponentiation grows rapidly with the modulus size, and thus there is a tradeoff between increasing the security of RSA and decreasing its efficiency.

It is also generally accepted that those moduli n consisting of prime factors p and q of roughly the same size are the hardest to factor.

—*General-purpose factoring methods*: Examples of such general factoring algorithms are the *general number field sieve* (see, e.g., [LL93]) or the older *quadratic sieve* (see, e.g., [Buc01,Sti95]). They are based on the following simple idea. Suppose n is the number to be factorized. Using the respective "sieve," one

determines integers a and b such that

$$a^2 \equiv b^2 \quad \mathrm{mod}\ n \ \text{ and } \ a \not\equiv \pm b \quad \mathrm{mod}\ n. \tag{2.8}$$

Thus, n divides $a^2 - b^2 = (a-b)(a+b)$, but neither $a-b$ nor $a+b$. Hence, $\gcd(a - b, n)$ is a nontrivial factor of n. The general number field sieve and the quadratic sieve differ in the specific way the integers a and b satisfying Equation (2.8) are found.

—*Special-purpose factoring methods*: Depending on the form of the primes p and q, it might be argued that using special-purpose factoring methods such as Pollard's "$p - 1$ method" [Pol74] may be more effective and more successful than using general-purpose factoring methods. This potential threat led to the introduction of *strong primes* that resist such special-purpose factoring methods. A strong prime p is required to satisfy certain conditions such as that $p - 1$ has a large factor r and $r - 1$, in turn, has a large factor, etc.

—*Elliptic curve method*: This factoring method was introduced by Lenstra [Len87], and it has some success probability regardless of the form of the primes chosen. Consequently, the most effective countermeasure against the elliptic curve method is to use primes of very large size. This countermeasure simultaneously provides, with a very high probability, protection against all known types of special-purpose factoring methods. In short, randomly chosen large primes are more important than strong primes. Note that weak primes are believed to be rare; Pomerance and Sorenson [PS95] study the density of weak primes.

—*Factoring on a quantum computer*: Last, we mention that Shor's algorithm for factoring large numbers on a quantum computer [Sho97] poses a potential threat to the security of RSA and other cryptosystems whose security relies on the hardness of the factoring problem. More precisely, Shor's efficient quantum algorithm determines the order of a given group element, a problem closely related to the factoring problem. Using Miller's randomized reduction [Mil76], if one can efficiently compute the order of group elements, then one can efficiently solve the factoring problem. However, the quantum computer is a theoretical construct currently. Whether or not Shor's quantum factoring algorithm will be a practical threat remains to be seen in the future.

Superencryption:. Early on Simmons and Norris [SN77] proposed an attack on RSA called superencryption. This attack is based on the observation that a sufficient number of encryptions will eventually recover the original message, since the RSA encryption function is an injective mapping onto a finite set, which makes the graph of the function a union of disjoint cycles. This attack is a threat to the security of RSA, provided that the number of encryptions required is small. Luckily, superencryption is not a practical attack if the primes are large and are chosen at random.

Wiener's attack:. Wiener [Wie90] proposed an attack on the RSA system by a continued fraction approximation, using the public key (n, e) to provide sufficient information to recover the private key d. More precisely, Wiener proved that if the keys in the RSA system are chosen such that $n = pq$, where $q < p < 2q$, and $d < \frac{1}{3} \sqrt[4]{n}$, then given the public key (n, e) with $ed \equiv 1 \quad \mathrm{mod}\ \phi(n)$ the private key d can be computed in linear time.

Here is a proof sketch of Wiener's result (see [Bon99]). Since $ed \equiv 1 \mod \phi(n)$, there exists a k such that $ed - k\phi(n) = 1$, which implies that $\frac{k}{d}$ is an approximation of $\frac{e}{\phi(n)}$:

$$\left| \frac{e}{\phi(n)} - \frac{k}{d} \right| = \left| \frac{1}{d\phi(n)} \right|. \tag{2.9}$$

Erich does not know $\phi(n)$, but he can use n in place of $\phi(n)$. Using $ed - k\phi(n) = 1$ and the easily verified fact that $|n - \phi(n)| < 3\sqrt{n}$, in place of Equation (2.9) we now have

$$\left| \frac{e}{n} - \frac{k}{d} \right| = \left| \frac{1 - k(n - \phi(n))}{dn} \right| \leq \left| \frac{3k\sqrt{n}}{dn} \right| = \frac{3k}{d\sqrt{n}}.$$

Since $k\phi(n) = ed - 1 < ed$ and $e < \phi(n)$, we have $k < d < \frac{1}{3}\sqrt[4]{n}$. Hence,

$$\left| \frac{e}{n} - \frac{k}{d} \right| < \frac{1}{d\sqrt[4]{n}} < \frac{1}{2d^2}.$$

There are at most $\log n$ fractions $\frac{k}{d}$ with $d < n$ approximating $\frac{e}{n}$ so tightly, and they can be obtained by computing the $\log n$ convergents of the continued fraction expansion of $\frac{e}{n}$ (see [HW79, Thm. 177]). Since $ed - k\phi(n) = 1$, we have $\gcd(k, d) = 1$, so $\frac{k}{d}$ is a reduced fraction.

Note that this attack is efficient and practical, and thus is a concern, only if the private key d is chosen to be small relative to n. For example, if n is a 1024 bits number, then d must be at least 256 bits long in order to prevent Wiener's attack. A small value of d, however, enables fast decryption and in particular is desirable for low-power devices such as "smartcards." Therefore, Wiener proposed certain techniques that avoid his attack.

The first technique is to use a large encryption exponent, say $\tilde{e} = e + \ell\phi(n)$ for some large ℓ. For a large enough $\tilde{e}$, the factor k in the above proof is so large that Wiener's attack cannot be mounted, regardless of how small d is.

The second technique uses the Chinese Remainder Theorem to speed up decryption, even if d is not small. Let d be a large decryption exponent such that both $d_p \equiv d \mod p - 1$ and $d_q \equiv d \mod q - 1$ are small. Then, one can decrypt a given cipher text c as follows. Compute $m_p = c^{d_p} \mod p$ and $m_q = c^{d_q} \mod q$, and use the Chinese Remainder Theorem to obtain the unique solution m modulo $n = pq$ of the two equations $m = m_p \mod p$ and $m = m_q \mod q$. The point is that although d_p and d_q are small, d can be chosen large enough to resist Wiener's attack.

Boneh and Durfee [BD00] recently improved Wiener's result: Erich can efficiently compute d from (n, e) provided that $d < n^{0.292}$.

Small-message attack:. RSA encryption is not effective if both the message m to be encrypted and the exponent e to be used for encryption are small relative to the modulus n. In particular, if $c = m^e < n$ is the cipher text, then m can be recovered from c by ordinary root extraction. Thus, either the public exponent should be large or the messages should always be large. It is this latter suggestion that is more useful, for a small public exponent is often preferred in order to speed up the encryption and to preclude Wiener's attack.

Low-exponent attack:. One should take precautions, though, not to choose the public exponent too small. A preferred value of e that has been used often in the past is $e = 3$. However, if three parties participating in the same system encrypt the same message m using the same public exponent 3, although perhaps different moduli n_1, n_2, and n_3, then one can easily compute m from the three cipher texts:

$$
\begin{aligned}
c_1 &= m^3 \mod n_1 \\
c_2 &= m^3 \mod n_2 \\
c_3 &= m^3 \mod n_3.
\end{aligned}
$$

In particular, the message m must be smaller than the moduli, and so m^3 will be smaller than $n_1 n_2 n_3$. Using the Chinese Remainder Theorem (see, e.g., [Knu81, Sti95]), one can compute the unique solution

$$
c = m^3 \mod n_1 n_2 n_3 = m^3.
$$

Hence, one can compute m from c by ordinary root extraction.

More generally, suppose that k related plain texts are encrypted with the same exponent e:

$$
\begin{aligned}
c_1 &= (a_1 m + b_1)^e \mod n_1 \\
c_2 &= (a_2 m + b_2)^e \mod n_2 \\
&\ \ \vdots \\
c_k &= (a_k m + b_k)^e \mod n_k,
\end{aligned}
$$

where a_i and b_i, $1 \le i \le k$, are known and $k > \frac{e(e+1)}{2}$ and $\min(n_i) > 2^{e^2}$. Then, an attacker can solve for m in polynomial time using lattice reduction techniques. This observation is due to Johan Håstad [Hås88], and his "broadcast attack" has been strengthened by Don Coppersmith [Cop97]. This attack is a concern if the messages are related in a known way. Padding the messages with pseudorandom strings prior to encryption prevents mounting this attack in practice, see, e.g., [KR95]. If the messages are related in a known way, they should not be encrypted with many RSA keys.

A recommended value of e that is commonly used today is $e = 2^{16} + 1$. One advantage of this value for e is that its binary expansion has only two ones, which implies that the square-and-multiply algorithm of Figure 2 requires very few operations,[4] and so is very efficient.

Forging RSA signatures:. This attack is based on the fact that the RSA encryption function is a homomorphism: if (n, e) is the public key and m_1 and m_2 are two messages then

$$
m_1^e \cdot m_2^e \equiv (m_1 \cdot m_2)^e \mod n. \tag{2.10}
$$

Another identity that can easily be verified is:

$$
(m \cdot r^e)^d \equiv m^d \cdot r \mod n. \tag{2.11}
$$

---

[4]How many exactly?

In particular, these identities can be used to mount an attack on the digital signature scheme based on the RSA algorithm, see Figure 4 and Section 2.3. Given previous message-signature pairs $(m_1, \mathrm{sig}_A(m_1)), \ldots, (m_k, \mathrm{sig}_A(m_k))$, Erich can use the congruences (2.10) and (2.11) to compute a new message-signature pair $(m, \mathrm{sig}_A(m))$ by

$$m = r^e \prod_{i=1}^{k} m_i^{e_i} \mod n;$$

$$\mathrm{sig}_A(m) = r \prod_{i=1}^{k} (\mathrm{sig}_A(m_i))^{e_i} \mod n,$$

where r and the e_i are arbitrary. Hence, Erich can forge Alice's signature without knowing her private key, and Bob will not detect the forgery, since $m \equiv (\mathrm{sig}_A(m))^e \mod n$. Note that, in Equation (2.10), even if m_1 and m_2 are meaningful plain texts, $m_1 \cdot m_2$ usually is not. Thus, Erich can forge Alice's signature only for messages that may or may not be useful. However, he might choose the messages m_i so as to generate a meaningful message m with a forged digital signature. This *chosen-plain-text attack* can again be avoided by pseudorandom padding techniques that destroy the algebraic relations between messages. Pseudorandom padding is also a useful countermeasure against the following *chosen-cipher-text attack*: Erich intercepts some cipher text c, chooses $r \in \mathbb{N}$ at random, and computes $c \cdot r^e \mod n$, which he sends to the legitimate receiver Bob. By Equation (2.11), Bob will decrypt the string $\hat{c} = c^d \cdot r \mod n$, which is likely to look like a random string. Erich, however, if he were to get his hands on $\hat{c}$, could obtain the original message m by multiplying by r^{-1}, the inverse of r modulo n, i.e., by computing $m = r^{-1} \cdot c^d \cdot r \mod n$.

3. PROTOCOLS FOR SECRET-KEY AGREEMENT, PUBLIC-KEY ENCRYPTION, AND DIGITAL SIGNATURES

Consider again a scenario where Alice and Bob want to exchange messages over an insecure channel such as a public telephone line, and where Erich is an eavesdropper:

Erich

Alice   **Bob**
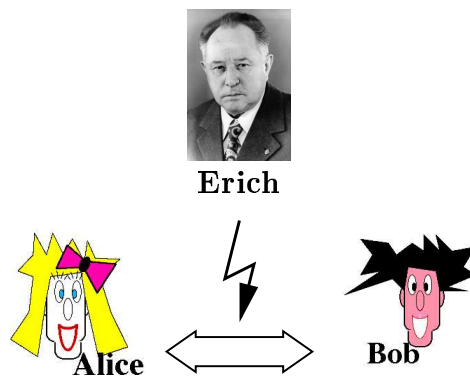
This is why Alice and Bob want to encrypt their messages. For efficiency purposes, they decide to use a symmetric cryptosystem in which they both possess the same

key for encryption and for decryption; recall Definition 1.1. But then, how can they agree on a joint secret key when they can communicate only over an insecure channel? If they were to send an encrypted message containing the key to be used in subsequent communications, which key should they use to encrypt *this* message?

This paradoxical situation is known as the *secret-key agreement* problem, and it was considered to be unsolvable since the beginning of cryptography. It was quite a surprise when in 1976 Whitfield Diffie and Martin Hellman [DH76] did solve this long-standing, seemingly paradoxical problem by proposing the first secret-key agreement protocol. We describe their protocol in Section 3.1. Interestingly, it was the Diffie–Hellman protocol that inspired Rivest, Shamir, and Adleman to invent the RSA system. That is, Diffie and Hellman's key idea to solve the secret-key agreement problem opened the door to modern public-key cryptography, which no longer requires sending secret keys over insecure channels.

Strangely enough, the reverse happened in the nonpublic sector. The Communications Electronics Security Group (CESG) of the British Government Communications Head Quarters (GCHQ) claims to have invented the RSA public-key cryptosystem prior to Rivest, Shamir, and Adleman and the Diffie–Hellman secret-key agreement scheme independently of Diffie and Hellman. And they did so in reverse order. James Ellis first discovered the principle possibility of public-key cryptography in the late sixties. In 1973, Clifford Cocks developed the mathematics necessary to realize Ellis's ideas and formulated what four years later became known as the RSA system. Soon thereafter, inspired by Ellis's and Cocks's work, Malcolm Williamson invented what became known as the Diffie–Hellman secret-key agreement scheme, around the same time Diffie and Hellman succeeded. None of the results of Ellis, Cocks, and Williamson became known to the public then. The full story—or what of it is publicly known by now—is told in Singh's book [Sin99].

Section 3.2 shows how to modify the Diffie–Hellman protocol in order to obtain a public-key cryptosystem. This protocol is due to Taher ElGamal [ElG85]. Just like the Diffie–Hellman protocol, ElGamal's cryptosystem is based on the difficulty of computing discrete logarithms.

Section 3.3 gives an interesting protocol due to an unpublished work of Adi Shamir. In this protocol, keys do not need to be agreed upon prior to exchanging encrypted messages.

Another cryptographic task is the generation of *digital signatures*: Alice wants to sign her encrypted messages to Bob in a way that allows Bob to verify that Alice was indeed the sender of the message. Digital signature protocols are used for the authentication of documents such as email messages. The goal is to preclude Erich from forging Alice's messages and her signature. Digital signature protocols are described in Section 2.3 (RSA digital signatures), in Section 3.2 (ElGamal digital signatures) and in Section 3.4 (Rabi and Sherman digital signatures).

3.1   Diffie and Hellman's Secret-Key Agreement Protocol

Figure 5 shows how the Diffie–Hellman secret-key agreement protocol works. It is based on the modular exponential function with base g and modulus p, where p is a prime and g is a primitive root of p in $\mathbb{Z}_p^*$, the cyclic group of prime residues modulo p; recall that $\mathbb{Z}_p^*$ has order $\phi(p) = p - 1$. The formal definition is as follows.

Step	Alice		Bob
1	Alice and Bob agree upon a large prime p and a primitive root g of p; p and g are public		
2	chooses a large number a at random, computes $\alpha = g^a \bmod p$		chooses a large number b at random, computes $\beta = g^b \bmod p$
3		$\xrightarrow{\alpha}$ $\xleftarrow{\beta}$	
4	computes her key $k_A = \beta^a \mod p$		computes his key $k_B = \alpha^b \mod p$

Fig. 5.   The Diffie–Hellman secret-key agreement protocol.

Definition 3.1. —For $n \in \mathbb{N}$, a *primitive root of n* is any element $a \in \mathbb{Z}_n^*$ satisfying that, for each d with $1 \leq d < \phi(n)$, it holds that

$$a^d \not\equiv 1 \mod n.$$

Equivalently, a primitive root of n is a generator of $\mathbb{Z}_n^*$.

—Let p be a prime, and let g be a primitive root of p. The function $\alpha_{(g,p)} : \mathbb{Z}_{p-1} \to \mathbb{Z}_p^*$ that is defined by

$$\alpha_{(g,p)}(a) = g^a \mod p.$$

is called the *modular exponential function with base g and modulus p*. Its inverse function, which for fixed p and g maps $\alpha_{(g,p)}(a)$ to $a = \log_g \alpha \mod p$, is called the *discrete logarithm*.

As noted above, every primitive root of p generates the entire group $\mathbb{Z}_p^*$. Moreover, $\mathbb{Z}_p^*$ has precisely $\phi(p-1)$ primitive roots. For example, $\mathbb{Z}_5^* = \{1, 2, 3, 4\}$ and $\mathbb{Z}_4^* = \{1, 3\}$, so $\phi(4) = 2$, and the two primitive roots of 5 in $\mathbb{Z}_5^*$ are 2 and 3, since

$$2^1 = 2; \quad 2^2 = 4; \qquad\quad 2^3 \equiv 3 \mod 5; \quad 2^4 \equiv 1 \mod 5;$$
$$3^1 = 3; \quad 3^2 \equiv 4 \mod 5; \quad 3^3 \equiv 2 \mod 5; \quad 3^4 \equiv 1 \mod 5.$$

Not every integer has a primitive root: 8 is the smallest such example. It is known from elementary number theory that an integer n has a primitive root if and only if n is 1 or 2 or 4, or is of the form q^k or $2q^k$ for some odd prime q.

The protocol from Figure 5 works, since

$$k_A = \beta^a = g^{ba} = g^{ab} = \alpha^b = k_B.$$

Thus, the keys computed by Alice and Bob indeed are the same.

Computing discrete logarithms is considered to be a very hard problem: no efficient algorithms are known for solving it. In contrast, the modular exponential function can be computed efficiently, using the fast exponentiation algorithm

"square-and-multiply" described as Figure 2. That is why modular exponentiation is considered to be a candidate for a "one-way function," i.e., a function that is easy to compute but hard to invert. Things are bad. It is currently not known whether or not one-way functions exist. Things are worse. Although they are not known to exist, one-way functions play a key role in cryptography, and the security of many cryptosystems is based on the assumption that one-way functions do exist. We will discuss the notion of one-way functions in more detail in Section 5.

If Erich is listening carefully to Alice and Bob's communication in the Diffie–Hellman protocol (see Figure 5), he knows p, g, α, and β. He wants to compute their joint secret key, $k_A = k_B$. This problem is known as the *Diffie–Hellman problem*. If Erich could solve the discrete logarithm problem efficiently, he could easily compute $a = \log_g \alpha \mod p$ and $b = \log_g \beta \mod p$ and, thus, $k_A = \beta^a \mod p$ and $k_B = \alpha^b \mod p$. That is, the Diffie–Hellman problem is no more difficult than the discrete logarithm problem. The converse question—of whether the Diffie–Hellman problem is as hard as the discrete logarithm problem—is still an unproven conjecture. Fortunately, as noted above, the discrete logarithm problem is viewed as being intractable, so this attack is very unlikely to be a practical threat. On the other hand, it is the only known attack for computing the keys directly from α and β in the Diffie–Hellman protocol. Note, however, that no proof of security for this protocol has been established up to date.

Note also that computing the keys $k_A = k_B$ directly from α and β is not the only possible attack on the Diffie–Hellman protocol. For example, it is vulnerable to the *Man-in-the-middle attack*. Unlike passive attacks against the underlying mathematics of a cryptosystem, in which an eavesdropper tries to gain information without affecting the protocol, the Man-in-the-middle attack is an active attack, in which an eavesdropper attempts to alter the protocol to his own advantage. That is, Erich, as the "man in the middle," might pretend to be Alice when communicating with Bob, and he might pretend to be Bob when communicating with Alice. He could intercept $\alpha = g^a \mod p$ that Alice sends to Bob and he could also intercept $\beta = g^b \mod p$ that Bob sends to Alice, passing on his own values α_E in place of α to Bob and β_E in place of β to Alice. That way Erich could compute two (possibly distinct) keys, one for communicating with Alice, the other one for communicating with Bob, without them having any clue that they in fact are communicating with him. Thus, Alice and Bob cannot be certain of the authenticity of their respective partners in the communication. In Section 4, we will introduce *zero-knowledge protocols*, which can be used to ensure proper authentication.

By slightly modifying the Diffie–Hellman protocol, it is possible to obtain a public-key cryptosystem. The variant of the Diffie–Hellman protocol presented here in fact is a "hybrid cryptosystem," a public-key cryptosystem making use of a given symmetric cryptosystem. Such hybrid systems are often useful in practice, for they combine the advantages of asymmetric and symmetric cryptosystems. Symmetric systems are usually more efficient than public-key systems.

The protocol works as follows. Alice and Bob agree on a large prime p and a primitive root g of p, which are public. They also agree on some symmetric cryptosystem $S = (\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ with encryption functions $\mathcal{E} = \{E_k \mid k \in \mathcal{K}\}$ and decryption functions $\mathcal{D} = \{D_k \mid k \in \mathcal{K}\}$. The subsequent steps of the protocol

Step	Alice		Bob
1	Alice and Bob agree upon a large prime p and a primitive root g of p; p and g are public		
2			chooses a large number b at random as his private key and computes $\beta = g^b \mod p$
3		$\xleftarrow{\beta}$	
4	chooses a large number a at random, computes $\alpha = g^a \mod p$, the key $k = \beta^a \mod p$, and the cipher text $c = E_k(m)$, where m is the message to be sent		
5		$\xrightarrow{\alpha, c}$	
6			computes $k = \alpha^b \mod p$ and $m = D_k(c)$

Fig. 6. A public-key cryptosystem based on the Diffie–Hellman protocol, which uses the encryption and decryption algorithms E_k and D_k of a given symmetric cryptosystem.

are shown in Figure 6. The message to be sent is encrypted using the symmetric system S, and the symmetric key k used in this encryption is transmitted in a Diffie–Hellman-like fashion. This modification of the original Diffie–Hellman protocol is the standard usage of Diffie–Hellman.

The system in Figure 6 modifies the original Diffie–Hellman protocol in the following way. While in the Diffie–Hellman scheme Alice and Bob *simultaneously* compute and send their "partial keys" α and β, respectively, they do so *sequentially* in the protocol in Figure 6. That is, Alice must wait for Bob's value β, his public key, to be able to compute the key k with which she then encrypts her message m via the symmetric cryptosystem S. Moreover, Bob generates, once and for all, his public β for possibly several communications with Alice, and also for possibly several users other than Alice who might want to communicate with him. In contrast, Alice has to generate her α anew again and again every time she communicates with Bob, just like in the original Diffie–Hellman protocol. This modification of Diffie–Hellman is usually referred to as Predistributed Diffie–Hellman. In a *key distribution scheme*, one party chooses a key and then transmits it to another party or parties over an insecure channel. In contrast, in a *secret-key agreement scheme* such as the original Diffie–Hellman protocol from Figure 5, two or more parties jointly compute, by communicating over an insecure channel, a shared secret key, which depends on inputs from both or all parties.

3.2 ElGamal's Public-Key Cryptosystem and Digital Signature Protocol

Taher ElGamal [ElG85] developed a public-key cryptosystem and a digital signature protocol that are based on the Diffie–Hellman protocol. In fact, the variant of Diffie–Hellman presented in Figure 6 is somewhat reminiscent of the original ElGamal public-key cryptosystem, which we will now describe.

Step	Alice		Bob
1	Alice and Bob agree upon a large prime p and a primitive root g of p; p and g are public		
2			chooses $b \in \mathbb{Z}^*_{p-1}$ at random and computes $\beta = g^b \mod p$; b is private and β is public
3		$\overset{\beta}{\Longleftarrow}$	
4	picks a secret $a \in \mathbb{Z}^*_{p-1}$ at random, computes $\alpha = g^a \mod p$ and $c = m\beta^a \mod p$, where m is the message to be sent		
5		$\overset{\alpha,\,c}{\Longrightarrow}$	
6			computes $x = p - 1 - b$ and decrypts by computing $m = c\alpha^x \mod p$

Fig. 7.   The ElGamal public-key cryptosystem.

Figure 7 shows ElGamal's public-key cryptosystem. After Alice and Bob have agreed on a prime p and a primitive root g of p, Bob picks a random value $b \in \mathbb{Z}^*_{p-1}$ and computes his public key $\beta = g^b \mod p$. If Alice wants to send him a message $m \in \mathbb{Z}^*_p$, she looks up β and "disguises" m by multiplying it with β^a modulo p, where $a \in \mathbb{Z}^*_{p-1}$ is a random number she has picked. This yields the first part c of the cipher text, the second part is $\alpha = g^a \mod p$. She sends both c and α to Bob. To decrypt, Bob first computes $x = p - 1 - b$. Since $1 \le b \le p - 2$, it follows that $1 \le x \le p - 2$. Bob then can recover the original plain text m by computing:

$$c\alpha^x \equiv m\beta^a g^{a(p-1-b)} \equiv mg^{ba+a(p-1)-ab} \equiv m \left(g^{p-1}\right)^a \equiv m \mod p.$$

Just as in the Diffie–Hellman protocol, the security of the ElGamal protocol is based on the difficulty of computing discrete logarithms. Although it is not known whether breaking the ElGamal protocol is as hard as solving the discrete logarithm problem, it can be shown that breaking the ElGamal protocol is precisely as hard as solving the Diffie–Hellman problem. To prevent known attacks on the ElGamal cryptosystem, the prime p should be chosen large enough (at least 150 digits long) and such that $p - 1$ has at least one large prime factor.

Step	Alice		Bob
1	Alice and Bob agree upon a large prime p and a primitive root g of p; p and g are public		
2			chooses b and $\beta = g^b \mod p$ as in Fig. 7; chooses a number r with $\gcd(r, p-1) = 1$, computes $\rho = g^r \mod p$ and s according to Eq. (3.12) and his signature $$\operatorname{sig}_B(m) = (\rho, s)$$
3		$\beta,\ m,\ \operatorname{sig}_B(m)$ $\Longleftarrow$	
4	verifies Bob's signature by checking that Eq. (3.13) holds: $$g^m \equiv \beta^\rho \cdot \rho^s \mod p.$$		

Fig. 8. The ElGamal digital signature protocol.

ElGamal's system can be modified so as to yield a digital signature protocol. A particularly efficient variant of this protocol that is due to an idea of Schnorr [Sch90] is now the United States "Digital Signature Standard" [Nat91,Nat92].

The ElGamal digital signature protocol is presented in Figure 8. Suppose that Bob wants to send a message m to Alice. To prove that he indeed is the sender, he wants to sign the message in a way that Alice can verify. Let a large prime p and a primitive root g of p be given as in the ElGamal public-key cryptosystem, see Figure 7. As in that protocol, Bob chooses his private b and computes $\beta = g^b$ mod p. In addition, he now chooses a number r coprime with $p-1$, and he computes $\rho = g^r \mod p$ and a solution s to the congruence

$$b \cdot \rho + r \cdot s \equiv m \mod p - 1 \qquad (3.12)$$

using the extended algorithm of Euclid, see Figure 1 and Lemma 2.3.

Bob keeps b and r secret, and he sends along with his message m his digital signature $\operatorname{sig}_B(m) = (\rho, s)$ and the value β to Alice.

Alice checks the validity of the signature by verifying the congruence

$$g^m \equiv \beta^\rho \cdot \rho^s \mod p. \qquad (3.13)$$

The protocol is correct, since by Fermat's Little Theorem (see Theorem 2.2) and by Equation (3.12), it holds that

$$g^m \equiv g^{b \cdot \rho + r \cdot s} \equiv \beta^\rho \cdot \rho^s \mod p.$$

Note that the public verification key, which consists of the values p, g, and β, is computed just once and can be used to verify any message that is signed with p, g, b, and β. However, a new value of r is chosen every time a message is signed.

3.3 Shamir's No-Key Protocol

Step	Alice		Bob
1	Alice and Bob agree upon a large prime p, which is public		
2	computes $x = m^a \mod p$, where m is the message		
3		$\overset{x}{\Longrightarrow}$	
4			computes $y = x^b \mod p$
5		$\overset{y}{\Longleftarrow}$	
6	computes $z = y^{a^{-1}} \mod p$		
7		$\overset{z}{\Longrightarrow}$	
8			computes $m = z^{b^{-1}} \mod p$

Fig. 9. Shamir's no-key protocol.

Adi Shamir proposed a cryptosystem by which Alice and Bob can exchange messages that are encrypted by Alice's and Bob's individual secret keys, yet in which there is no need for Alice and Bob to previously agree on a *joint* secret key. This clever idea is described in an unpublished paper of Shamir, and it is again based on the modular exponentiation function and the difficulty of efficiently computing discrete logarithms that was useful for the Diffie–Hellman secret-key agreement protocol described in Section 3.1. The Shamir protocol is often called Massey-Omura in the literature. Both inventors were preceded by Malcolm Williamson from GCHQ who developed the same protocol in the nonpublic sector around 1974.

Figure 9 shows how Shamir's no-key protocol works. In this protocol, let m be the message that Alice wants to send to Bob. First, Alice and Bob agree on a large prime p. Alice generates a pair (a, a^{-1}) satisfying

$$aa^{-1} \equiv 1 \mod p - 1,$$

where a^{-1} is the inverse of a modulo $p - 1$. Recall from Section 2.2 that, given a prime p and an integer $a \in \mathbb{Z}_p^*$, the inverse a^{-1} of a modulo $p - 1$ can easily be computed. Similarly, Bob generates a pair (b, b^{-1}) satisfying

$$bb^{-1} \equiv 1 \mod p - 1,$$

where b^{-1} is the inverse of b modulo $p - 1$. See Figure 9 for the rest of the steps.

The protocol is correct, since for all messages m, $1 \leq m \leq p$, it holds that:

$$m \equiv m^{aa^{-1}} \mod p \quad \text{and} \quad m \equiv m^{bb^{-1}} \mod p.$$

Hence, looking at Figure 9, we obtain

$$z^{b^{-1}} \equiv y^{a^{-1}b^{-1}} \equiv x^{ba^{-1}b^{-1}} \equiv m^{aba^{-1}b^{-1}} \equiv m \mod p,$$

so Step 8 of Figure 9 is correct.

Note that modular exponentiation is used here both for encryption and decryption. The key property for this protocol to work is that modular exponentiation is symmetric in the exponents, i.e., for all a and b, it holds that

$$\alpha_{(g,p)}(a \cdot b) \equiv g^{a \cdot b} \equiv g^{b \cdot a} \mod p.$$

3.4 Rivest, Rabi, and Sherman's Secret-Key Agreement and Digital Signature Protocols

Ron Rivest, Muhammad Rabi, and Alan Sherman developed secret-key agreement and digital signature protocols. The secret-key agreement protocol from Figure 10 is attributed to Rivest and Sherman in [RS93,RS97]. The digital signature protocol from Figure 11 is due to Rabi and Sherman [RS93,RS97].

Here is a brief, intuitive explanation of how these protocols work. The key building block of both protocols is a *total, strongly noninvertible, associative one-way function*. As mentioned earlier, one-way functions are theoretical constructs not known to exist. However, there are plausible assumptions under which one-way functions of various types can be constructed. In Section 5, under a quite plausible complexity-theoretic assumption, we will see how to construct a concrete candidate for a total, strongly noninvertible, associative one-way function. For now, assume that σ *is* such a function. That is, σ is a total two-ary (i.e., two-argument) function mapping pairs of positive integers to positive integers such that:

—σ is *associative*, i.e., the equation $\sigma(x, \sigma(y, z)) = \sigma(\sigma(x, y), z)$ holds for all $x, y, z \in \mathbb{N}$.

—σ is *strongly noninvertible*, i.e., σ is hard to invert even if in addition to the function value one of the arguments is given.

Look at Rivest and Sherman's secret-key agreement protocol in Figure 10. Since σ is associative, we have:

$$k_A = \sigma(x, \sigma(y, z)) = \sigma(\sigma(x, y), z) = k_B,$$

and thus the keys computed by Alice and Bob indeed are the same. On the other hand, if Erich was listening carefully, he knows not only two function values, $\sigma(x, y)$ and $\sigma(y, z)$, but he also knows y, the first argument of $\sigma(y, z)$ and the second argument of $\sigma(x, y)$. That is why σ must be strongly noninvertible, in order to prevent the direct attack that Erich computes Alice's secret number x from $\sigma(x, y)$ and y or Bob's secret number z from $\sigma(y, z)$ and y, in which case he could easily obtain their joint secret key, $k_A = k_B$. Analogous comments apply to Rabi and Sherman's digital signature protocol presented in Figure 11.

3.5 Discussion of Diffie–Hellman versus Rivest–Sherman

While the secret-key agreement protocol of Diffie and Hellman [DH76] is widely used in practice, that of Rivest and Sherman (see [RS93,RS97]) is not (yet) used in applications and, thus, might appear somewhat exotic at first glance. Note, however, that neither the Diffie–Hellman nor the Rivest–Sherman protocol has a proof of security up to date. So, let us digress for a moment to compare the state of the art on these two protocols.

Step	Alice		Bob
1	chooses two large numbers x and y at random, keeps x secret, and computes $\sigma(x,y)$		
2		$\xrightarrow{y,\ \sigma(x,y)}$	
3			chooses a large number z at random, keeps z secret and computes $\sigma(y,z)$
4		$\xleftarrow{\sigma(y,z)}$	
5	computes her key $k_A = \sigma(x,\sigma(y,z))$		computes his key $k_B = \sigma(\sigma(x,y),z)$

Fig. 10. The Rivest–Sherman secret-key agreement protocol, which uses a strongly noninvertible, associative one-way function σ.

Step	Alice		Bob
1	chooses two large numbers x_A and y_A at random, keeps x_A secret, and computes $\sigma(x_A,y_A)$		
2		$\xrightarrow{y_A,\ \sigma(x_A,y_A)}$	
3	computes her signature $\mathrm{sig}_A(m) = \sigma(m,x_A)$ for the message m		
4		$\xrightarrow{m,\ \mathrm{sig}_A(m)}$	
5			verifies Alice's signature by checking whether $\sigma(m,\sigma(x_A,y_A))$ equals $\sigma(\sigma(m,x_A),y_A)$

Fig. 11. The Rabi–Sherman digital signature protocol, which uses a strongly noninvertible, associative one-way function σ.

—While the Diffie–Hellman protocol uses a concrete function, the Rivest–Sherman protocol is based on an unspecified, "abstract" function that is described only by listing the properties it should satisfy. That is not to say that Rivest–Sherman is an abstract version of Diffie–Hellman. Rather, the Rivest–Sherman protocol may be seen as an alternative to the Diffie–Hellman protocol. The advantage of Rivest and Sherman's approach is that it is more flexible, as it does not depend on a single function.

—The security of the Diffie–Hellman scheme is based on the (unproven, yet plausible) assumption that computing discrete logarithms is a computationally intractable task.

In contrast, the Rivest–Sherman scheme uses a candidate for a strongly noninvertible, associative one-way function (see Section 5.1 for the formal definition) as its key building block. Although it is not known whether such functions exist, it has been shown recently by Hemaspaandra and this author [HR99] that they do exist in the worst-case model under the (unproven, yet plausible) assumption that $P \neq NP$, where P denotes the class of polynomial-time solvable problems, and NP denotes the class of problems that can be solved nondeterministically in polynomial time. Section 5 presents this result and a sketch of its proof.

—Breaking Diffie–Hellman is not even known to be as hard as computing discrete logarithms, even though some nice progress in this direction has been made recently by Maurer and Wolf [MW99], who established conditions for relating the hardness of breaking Diffie–Hellman to that of computing discrete logarithms. Again, their results rest on unproven, yet plausible assumptions. In particular, let $\nu(p)$ denote the minimum, taken over all numbers d in the interval $[p - 2\sqrt{p} + 1, p + 2\sqrt{p} + 1]$, of the largest prime factors of d. The "smootheness assumption" says that $\nu(p)$ is polynomial in $\log p$. Why is this assumption plausible? The idea is that numbers in the Hasse-Weil interval (which are sizes of elliptic curves) are smooth with the same probability as random numbers of the same length, and these probabilities are independent. Under this smoothness assumption, Maurer and Wolf [MW99] proved that breaking Diffie–Hellman and computing the discrete logarithm are polynomial-time equivalent tasks in the underlying cyclic group, where the equivalence is nonuniform.

Similarly, even if strongly noninvertible, associative one-way functions were known to exist, one could not conclude that the Rivest–Sherman protocol is secure; rather, strong noninvertibility merely precludes certain types of direct attacks [RS97,HR99]. Moreover, strongly noninvertible, associative one-way functions could be constructed so far only in the *worst-case* complexity model, assuming $P \neq NP$. Although this result is relevant and interesting in a complexity-theoretic setting, it has no direct implications in applied cryptography. For cryptographic applications, one would need to construct such functions based on the *average-case* complexity model, under plausible assumptions.

As noted in the outline of the tutorial, there is some hope for obtaining such a strong result by combining Hemaspaandra and Rothe's [HR99] technique on constructing strongly noninvertible, associative one-way functions in the worst case with Ajtai's [Ajt96] techniques on constructing hard instances of lattice problems. The shortest lattice vector problem, denoted by SVP, is the problem of finding a

shortest lattice vector in the lattice generated by a given lattice basis. Roughly speaking, Ajtai [Ajt96] proved that the problem SVP is as hard in the average-case as it is in the worst-case complexity model.

More precisely, Ajtai constructed an infinite family $\{\Lambda_n\}_{n \geq 1}$ of lattices, where each Λ_n is represented by a basis as an instance of SVP, and he showed the following result: Suppose one can compute in polynomial time, for each n, an approximately shortest vector in a lattice Λ_i *randomly* chosen from $\{\Lambda_n\}_{n \geq 1}$, with non-negligible probability. Then, the length of a shortest vector in *every* lattice from $\{\Lambda_n\}_{n \geq 1}$ can be estimated to within a fixed polynomial factor in polynomial time with probability close to one. However, since the best approximation factor known to be achieved by polynomial-time algorithms is essentially exponential, and since the best algorithms known to achieve polynomial-factor approximations run in exponential time, it follows that, as mentioned above, "SVP is as hard in the average-case as it is in the worst-case model." In this regard, the SVP is a unique problem; for no other problem in NP that is believed to be outside P such a strong connection is known to hold.

Based on the worst-case/average-case equivalence of SVP, Ajtai and Dwork [AD97] designed a public-key cryptosystem whose cryptographic security depends only on worst-case complexity assumptions. However, the worst-case hardness of SVP (in the Euclidean norm) had remained an open problem for a long time. Solving this problem, Ajtai [Ajt98] established the NP-hardness of SVP under randomized reductions. His result was strengthened by Micciancio [Mic01], who also simplified Ajtai's proof. Since the construction of strongly noninvertible, associative one-way functions in [HR99] is based on the assumption P $\neq$ NP, it seems reasonable to consider the NP-hard problem SVP to be a good candidate for achieving strongly noninvertible, associative one-way functions even in the technically more demanding average-case model.

The complexity of SVP and the use of lattices in crytography are covered in the surveys by Cai [Cai99], Kumar and Sivakumar [KS01], and Nguyen and Stern [NS01]. Interestingly, lattices are useful both in breaking existing cryptosystems like RSA (e.g., the low-exponent attacks of Håstad [Hås88] and Coppersmith [Cop97]), see Section 2.4) and in designing secure cryptosystems (e.g., the Ajtai-Dwork public-ley cryptosystem).

4. INTERACTIVE PROOF SYSTEMS AND ZERO-KNOWLEDGE PROTOCOLS

In Section 3.1, we mentioned the Man-in-the-middle attack on the Diffie–Hellman secret-key agreement protocol. Imagine that Bob has just agreed with his partner on a joint secret key via a public telephone line. Of course, he assumes it was Alice he was talking to. Bob was so clever to use the Diffie–Hellman protocol, and so he

thinks that Erich does not have a clue about what secret key they have chosen:
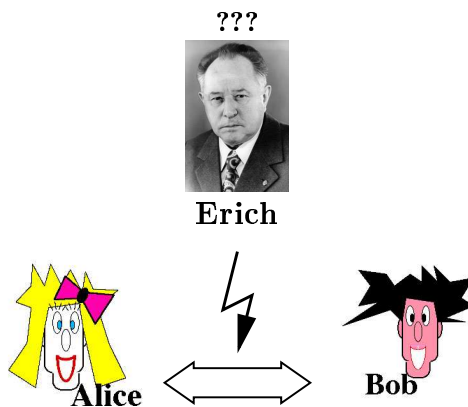
But Erich was even smarter. Here is what really happened:
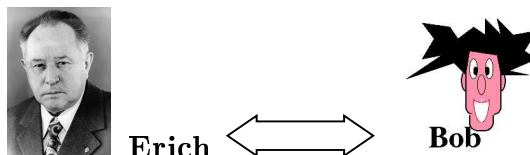
This situation raises the issue of *authentication*: How can Bob be certain that it in fact was Alice he was communicating with, and not Erich pretending to be Alice? In other words, how can Alice prove her identity to Bob beyond any doubt?

In Section 3, we have seen how to use digital signatures for the authentication of documents such as email messages. In this section, our goal is to achieve authentication of an *individual* rather than a document. One way to achieve this goal is to assign to Alice's identity some secret information such as her PIN (*"Personal Identifaction Number"*) or any other private information that nobody else knows. We refer to the information proving Alice's identity as Alice's *secret*.

But here's another catch. Alice would like to convince Bob of her identity by proving that she knows her secret. Ideally, however, she should not disclose her secret because then it wouldn't be a secret anymore: If Bob, for example, knew Alice's secret, he could pretend to be Alice when communicating with somebody else. So the question is:

How can one prove the knowledge of a secret without telling the secret?

That is precisely what zero-knowledge protocols are all about.

4.1 Interactive Proof Systems

Zero-knowledge protocols are a special form of interactive proof systems, which we will describe first. Interactive proof systems were introduced by Shafi Goldwasser, Silvio Micali, and Charles Rackoff [GMR85,GMR89]. Independently, Babai and Moran [BM88,Bab85] developed the essentially equivalent notion of Arthur-Merlin games.

As in the previous protocols, we consider the communication between two parties, the "prover" Alice and the "verifier" Bob:
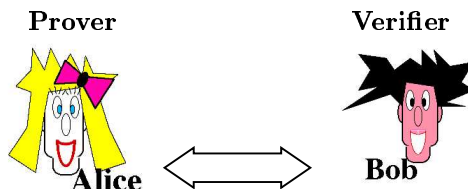
For now, we are not interested in the security aspects that may arise when the communication is eavesdropped; rather, we are concerned with the following communication problem: Alice and Bob want to jointly solve a given problem L, i.e., they want to decide whether or not any given instance belongs to L. For concreteness, consider the graph isomorphism problem.

Definition 4.1. The vertex set of any graph G is denoted by $V(G)$, and the edge set of G is denoted by $E(G)$. Let G and H be undirected, simple graphs, i.e., graphs with no reflexive or multiple edges.

An *isomorphism* between G and H is a bijective mapping π from $V(G)$ onto $V(H)$ such that, for all $i, j \in V(G)$,

$$\{i, j\} \in E(G) \iff \{\pi(i), \pi(j)\} \in E(H).$$

Graph-Isomorphism denotes the set of all pairs of isomorphic graphs.

The graph isomorphism problem is to determine whether or not any two given graphs are isomorphic. This problem belongs to NP, and since there is no efficient algorithm known for solving it, it is widely considered to be a hard, intractable problem. However, it is not known to be complete for NP, i.e., it is not known whether this problem belongs to the hardest NP problems. In fact, due to its "lowness" properties, it is doubted that the graph isomorphism problem is NP-complete. A set A is low for a complexity class $\mathcal{C}$ if it does not yield any additional computational power when used as an oracle by the machines representing the class $\mathcal{C}$, i.e., if $\mathcal{C}^A = \mathcal{C}$. Schöning [Sch87] showed that Graph-Isomorphism is in the second level of the low hierarchy within NP, i.e., it is low for $\mathrm{NP}^{\mathrm{NP}}$, the second level of the polynomial hierarchy. It follows that if Graph-Isomorphism were NP-complete then the polynomial hierarchy would collapse, which is considered unlikely. Moreover, Köbler et al. [KST92] proved Graph-Isomorphism low for PP, probabilistic polynomial time.

Therefore, it is conjectured that the graph isomorphism problem might be neither in P nor NP-complete, and this is what makes this problem so interesting for complexity theoreticians. Of course, proving this conjecture would immediately prove P different from NP; so, such a proof seems beyond current techniques. For more complexity-theoretic background on the graph isomorphism problem, we refer to the book by Köbler, Schöning, and Torán [KST93].

We mention in passing that (language versions of) the factoring problem and the discrete logarithm problem are not known to be NP-complete either. Unlike the graph isomorphism problem, however, no lowness properties are known for these two problems. Grollmann and Selman [GS88] have shown that a language version

of the discrete logarithm problem is contained in UP, which denotes Valiant's class "unambiguous polynomial time" [Val76]. NP-complete problems are very unlikely to belong to UP, so this result gives some evidence against the NP-completeness of the discrete logarithm problem.

Returning to Alice and Bob's communication problem, their task is to decide whether or not any given pair (G, H) of graphs is isomorphic. Alice, the prover, tries to *prove* them isomorphic by providing Bob with an isomorphism π between G and H. She intends to convince Bob *no matter whether or not G and H in fact are isomorphic*. But Bob is impatient. To accept the input, he wants to be convinced with overwhelming probability that the proof provided by Alice indeed is correct. Even worse, he is convinced only if *every potential prover strategy* Alice might come up with yields an overwhelming success probability. If Alice can accomplish this then Bob accepts the input, otherwise he rejects it.

To formalize this intuition, imagine Alice and Bob to be Turing machines. Alice, the prover, is an all-powerful Turing machine with no computational limitation whatsoever. Bob, the verifier, is a randomized Turing machine working in polynomial time, but capable of making random moves by flipping an unbiased coin. In Definition 4.2 below, in case of acceptance, it is enough that Alice finds one sufficient strategy to convince Bob. In case of rejection, however, rather than considering every potential prover strategy of Alice, it is useful to quantify over all possible provers that may replace Alice.

For the definition of randomized Turing machines, we refer to any textbook on complexity theory such as [BDG95,BC93,HO02,Pap94,Pap94]. Essentially, every nondeterministic Turing machine can be viewed as a randomized Turing machine by defining a suitable probability measure on the computation trees of the machine.

Definition 4.2 Interactive Proof System. [GMR85,GMR89]

(1) An *interactive proof system* (or "*IP protocol*") (A, B) is a protocol between Alice, the prover, and Bob, the verifier. Alice runs a Turing machine A with no limit on its resources, while Bob runs a polynomial-time randomized Turing machine B. Both access the same input on a joint input tape, and they are equipped with private work tapes for internal computations. They also share a read-write communication tape to exchange messages. Alice does not see Bob's random choices. Let $\Pr((A, B)(x) = 1)$ denote the probability (according to the random choices made in the communication) that Bob accepts the input x; i.e., for a particular sequence of random bits, "$(A, B)(x) = 1$" denotes the event that Bob is convinced by Alice's proof for x and accepts.

(2) An *interactive proof system* (A, B) *accepts a set L* if and only if for each x:

$$x \in L \implies (\exists A)\,[\Pr((A, B)(x) = 1) \geq \tfrac{3}{4}]; \tag{4.14}$$

$$x \notin L \implies (\forall \widehat{A})\,[\Pr((\widehat{A}, B)(x) = 1) \leq \tfrac{1}{4}], \tag{4.15}$$

where in (4.14) we quantify over the prover strategies (or "proofs") for x of the prescribed Turing machine A, whereas in (4.15) we quantify over the proofs $\widehat{A}$ for x of any prover (i.e., any Turing machine of unlimited computational power) that may replace the fixed Turing machine A.

(3) IP denotes the class of all sets that can be accepted by an interactive proof system.

Note that the acceptance probabilities of at least $\frac{3}{4}$ if $x \in L$ (respectively, of at most $\frac{1}{4}$ if $x \notin L$) are chosen at will. By probability amplification techniques [Pap94, BDG95,BC93], one can use any constants $\frac{1}{2} + \epsilon$ and $\frac{1}{2} - \epsilon$, respectively, where $\epsilon > 0$. It is even possible to make the error probability as small as $2^{-p(|x|)}$, for any fixed polynomial p. Better yet, Goldreich, Mansour, and Sipser [GMS87] have shown that one can even require the acceptance probability of exactly 1 if $x \in L$, without changing the class IP.

In the literature, verifier and prover are sometimes referred to as *Arthur* and *Merlin*. In fact, the Arthur-Merlin games introduced by Babai and Moran [BM88, Bab85] are nothing else than the interactive proof systems of Goldwasser et al. [GMR85,GMR89]. One difference between Definition 4.2 and the definition of Arthur-Merlin games is that the random bits chosen by Arthur are public (i.e., they are known to Merlin), while they are private to Bob in Definition 4.2. However, Goldwasser and Sipser [GS89] have shown that the privacy of the verifier's random bits does not matter: Arthur-Merlin games are equivalent to interactive proof systems.

What if Bob has run out of coins? That is, what if he behaves deterministically when verifying Alice's proof for "$x \in L$"? Due to her unlimited computational power, Alice can provide proofs of unlimited length, i.e., of length not bounded by any function in the length of x. However, since Bob is a polynomial-time Turing machine, it is clear that he can check only proofs of length polynomially in $|x|$. It follows that IP, when restricted to deterministic polynomial-time verifiers, is just a cumbersome way of defining the class NP. Hence, since `Graph-Isomorphism` belongs to NP, it must also belong to the (unrestricted) class IP. We omit presenting an explicit IP protocol for `Graph-Isomorphism` here, but we refer to Section 4.3, where in Figure 13 an IP protocol for `Graph-Isomorphism` with an additional property is given: it is a zero-knowledge protocol.

But what about the complement of `Graph-Isomorphism`? Does there exist an interactive proof system that decides whether or not two given graphs are *non*-isomorphic? Note that even though Alice is all-powerful computationally, she may run into difficulties when she is trying to prove that the graphs are non-isomorphic. Consider, for example, two non-isomorphic graphs with 1000 vertices each. A proof of that fact seems to require Alice to show that none of the 1000! possible permutations is an isomorphism between the graphs. Not only would it be impossible for Bob to check such a long proof in polynomial time, also for Alice it would be literally impossible to write this proof down. After all, 1000! is approximately $4 \cdot 10^{2567}$. This number exceeds the number of atoms in the entire visible universe,[5] which is currently estimated to be around 10^{77}, by a truly astronomical factor.

That is why the following result of Goldreich, Micali, and Wigderson [GMW86, GMW91] was a bit of a surprise.

THEOREM 4.3. [GMW86,GMW91]   `Graph-Isomorphism` *is in* IP.

---

[5]Dark matter excluded.

PROOF. Figure 12 shows the interactive proof system for the graph non-isomorphism problem.

Step	Alice		Bob
Input: Two graphs G_1 and G_2			
1			randomly chooses a permutation π on $V(G_1)$ and a bit $b \in \{1, 2\}$, and computes $H = \pi(G_b)$
2		$\overset{H}{\Longleftarrow}$	
3	determines $a \in \{1, 2\}$ such that G_a and H are isomorphic		
4		$\overset{a}{\Longrightarrow}$	
5			accepts if and only if $a = b$

Fig. 12.   The Goldreich-Micali-Wigderson IP protocol for $\overline{\texttt{Graph-Isomorphism}}$.

Let us check that the implications (4.14) and (4.15) from Definition 4.2 do hold. Suppose that G_1 and G_2 are non-isomorphic. Then, it is easy for Alice to determine that graph G_b, $b \in \{1, 2\}$, to which H is isomorphic. So she sends $a = b$, and Bob accepts with probability 1. That is,

$$(G_1, G_2) \in \overline{\texttt{Graph-Isomorphism}} \implies (\exists A)\, [\Pr((A, B)(G_1, G_2) = 1) = 1].$$

Now suppose that G_1 and G_2 are isomorphic. Then, no matter what clever strategy Alice applies, her chance of answering correctly (i.e., with $a = b$) is no better than $\frac{1}{2}$ because she does not see Bob's random bit b and so can do no better than guessing. That is,

$$(G_1, G_2) \notin \overline{\texttt{Graph-Isomorphism}} \implies (\forall \widehat{A})\, [\Pr((\widehat{A}, B)(G_1, G_2) = 1) \leq \frac{1}{2}].$$

Note that the acceptance probability of $\leq \frac{1}{2}$ above is not yet the acceptance probability of $\leq \frac{1}{4}$ required in (4.15) of Definition 4.2. However, as mentioned above, standard probability amplification techniques yield an error probability as close to zero as one desires. We leave the details to the reader. ∎

By definition, IP contains all of NP. The above result shows that IP also contains a problem from coNP, the class of complements of NP problems, which is unlikely to be contained in NP. So, the question arises of how big the class IP actually is. A famous result of Adi Shamir [Sha92] settled this question: IP equals PSPACE, the class of problems that can be decided in polynomial space.

4.2 Zero-Knowledge Protocols

Recalling the issue of authentication mentioned at the beginning of this section, we are now ready to define zero-knowledge protocols.

As mentioned above, `Graph-Isomorphism` is in IP. To prove that the two given graphs are isomorphic, Alice simply sends an isomorphism π to Bob, which he then checks deterministically in polynomial time. Suppose, however, that Alice wants to keep the isomorphism π secret. On the one hand, she does not want to disclose her secret; on the other hand, she wants to prove to Bob that she knows it. What she needs is a very special IP protocol that conveys nothing about her secret isomorphism, and yet proves that the graphs are isomorphic. The next section will present such a zero-knowledge protocol for `Graph-Isomorphism`.

But what is a zero-knowledge protocol and how can one formalize it? The intuition is this. Imagine that Alice has a twin sister named Malice who looks just like her. However, Malice does not know Alice's secret. Moreover, Malice does not have Alice's unlimited computational power; rather, just as the verifier Bob, she only operates like a randomized polynomial-time Turing machine. Still, she tries to simulate Alice's communication with Bob. An IP protocol has the *zero-knowledge property* if the information communicated in Malice's simulated protocol cannot be distinguished from the information communicated in Alice's original protocol. Malice, not knowing the secret, cannot put any information about the secret into her simulated protocol, and yet she is able to generate that clone of the original protocol that looks just like the original to an independent observer. Consequently, the verifier Bob (or any other party such as Erich) cannot extract any information from the original protocol. In short, if there's nothing in there, you can't get anything out of it.

Definition 4.4 *Zero-Knowledge Protocols.* [GMR85,GMR89]   Let (A, B) be an interactive proof system accepting a problem L. We say (A, B) is a *zero-knowledge protocol for L* if and only if there exists a simulator Malice such that the following holds:

—Malice runs a randomized polynomial-time Turing machine M to simulate the prover Alice in her communication with Bob, thus yielding a simulated protocol (M, B);

—for each $x \in L$, the tuples $(a_1, a_2, \ldots, a_k)$ and $(m_1, m_2, \ldots, m_k)$ representing the communication in (A, B) and in (M, B), respectively, are identically distributed over the coin tosses of A and B in (A, B) and of M and B in (M, B), respectively.

The above definition is called "honest-verifier perfect zero-knowledge" in the literature. That is, (a) one assumes that the verifier is *honest*, and (b) one requires that the information communicated in the simulated protocol *perfectly* coincides with the information communicated in the original protocol.

Assumption (a) is not quite realistic for most cryptographic applications. A dishonest verifier might alter the protocol to his own advantage. Therefore, one should modify the definition above to require that for *each* verifier B^* there exists a simulator M^* generating a simulated protocol not distinguishable from the original one. However, honest-verifier zero-knowledge protocols with public random bits can always be transformed to protocols that have the zero-knowledge property also

in the presence of dishonest verifiers.

Regarding assumption (b), there are several other notions of zero-knowledge that are weaker than perfect zero-knowledge, such as "statistical zero-knowledge" and "computational zero-knowledge." In a *statistical zero-knowledge protocol* (also known as *almost-perfect zero-knowledge protocol*), one requires that the information communicated in the original and in the simulated protocol be indistinguishable by certain statistical tests. In a *computational zero-knowledge protocol*, one merely requires that the information communicated in the original and in the simulated protocol be computationally indistinguishable, i.e., for each randomized polynomial-time Turing machine, the probability of detecting differences in the corresponding distributions is negligibly small.

In the latter model, Goldreich, Micali, and Wigderson [GMW86,GMW91] showed what is considered by far the most important result on zero-knowledge: Every problem in NP has a computational zero-knowledge protocol under the plausible assumption that there exist cryptographically secure bit-commitment schemes. The key idea is a computational zero-knowledge protocol for `Graph-Three-Colorability`, a well-known NP-complete problem. In contrast, it seems unlikely [BC89] that such a strong claim can be proven for the perfect zero-knowledge model presented in Definition 4.4.

For more information about interactive proof systems and zero-knowledge, we refer to the books by Goldreich [Gol01b, Chapter 4], Köbler et al. [KST93, Chapter 2], Papadimitriou [Pap94, Chapter 12.2], Balcázar et al. [BDG90, Chapter 11], and Bovet et al. [BC93, Chapter 10] and to the surveys by Oded Goldreich [Gol88], Shafi Goldwasser [Gol89], and Joan Feigenbaum [Fei92].

4.3 Zero-Knowledge Protocol for the Graph Isomorphism Problem

Oded Goldreich, Silvio Micali, and Avi Wigderson [GMW86,GMW91] proposed a zero-knowledge protocol for the graph isomorphism problem. This result was quite a surprise, since previously zero-knowledge protocols were known only for problems contained both in NP and coNP. It is considered to be unlikely that NP equals coNP; in particular, it is considered to be unlikely that `Graph-Isomorphism` is in coNP.

THEOREM 4.5. [GMW86,GMW91] `Graph-Isomorphism` *has a zero-knowledge protocol.*

PROOF. Figure 13 shows the Goldreich-Micali-Wigderson protocol. One difference to the protocol for the graph non-isomorphism problem in Figure 12 is that now Alice too makes random choices.

Alice's secret is the isomorphism π she has chosen. The protocol is correct, since Alice knows her secret π and also her random permutation ρ. Hence, she can easily compute the isomorphism σ with $\sigma(G_b) = H$ to prove her identity to Bob. When doing so, she does not have to disclose her secret π to Bob in order to convince him of her identity. In particular,

$$(G_1, G_2) \in \text{Graph-Isomorphism} \implies (\exists A)\,[\Pr((A, B)(G_1, G_2) = 1) = 1],$$

so the implication (4.14) from Definition 4.2 holds. Since Alice herself has chosen two isomorphic graphs, the case $(G_1, G_2) \notin \text{Graph-Isomorphism}$ does not oc-

cur, so the implication (4.15) from Definition 4.2 trivially holds if the protocol is implemented properly. Thus, the protocol is an interactive proof system for Graph-Isomorphism.

Recall that Alice wants to prove her identity via this protocol. Suppose that Erich or Malice want to cheat by pretending to be Alice. They do not know her secret isomorphism π, but they do know the public isomorphic graphs G_1 and G_2. They want to convince Bob that they know Alice's secret, which corresponds to (G_1, G_2). If, by coincidence, Bob's bit b equals their previously chosen bit a, they win. However, if $b \neq a$, computing $\sigma = \rho \circ \pi$ or $\sigma = \rho \circ \pi^{-1}$ requires knowledge of π. Without knowing π, computing π from the public graphs G_1 and G_2 seems to be impossible for them, since Graph-Isomorphism is a hard problem, too hard even for randomized polynomial-time Turing machines. Thus, they will fail provided that the graphs are chosen large enough.

Step	Alice		Bob
	Generation of isomorphic graphs and a secret isomorphism		
1	chooses a large graph G_1, a random permutation π on G_1's vertices, and computes the graph $G_2 = \pi(G_1)$; (G_1, G_2) are public, π is private		
	Protocol		
2	randomly chooses a permutation ρ on $V(G_1)$ and a bit $a \in \{1, 2\}$, computes $H = \rho(G_a)$		
3		$\xRightarrow{\ H\ }$	
4			chooses a bit $b \in \{1, 2\}$ at random and wants to see an isomorphism between G_b and H
5		$\xLeftarrow{\ b\ }$	
6	computes the permutation $$\sigma = \begin{cases} \rho & \text{if } b = a \\ \rho \circ \pi & \text{if } 1 = b \neq a = 2 \\ \rho \circ \pi^{-1} & \text{if } 2 = b \neq a = 1 \end{cases}$$ satisfying $\sigma(G_b) = H$		
7		$\xRightarrow{\ \sigma\ }$	
8			verifies that indeed $$\sigma(G_b) = H$$ and accepts accordingly

Fig. 13. The Goldreich-Micali-Wigderson zero-knowledge protocol for graph isomorphism.

Since they cannot do better than guessing the bit b, they can cheat with probability at most $\frac{1}{2}$. Of course, they can always guess the bit b, which implies that their chance of cheating successfully is exactly $\frac{1}{2}$. Hence, if Bob demands, say, k independent rounds of the protocol to be executed, he can make the cheating probability as small as 2^{-k}, and thus is very likely to detect any cheater. Note that after only 20 rounds the odds of malicious Malice getting away with it undetected are less than one to one million. Hence, the protocol is correct.

Step	Malice		Bob
	Simulated generation of isomorphic graphs		
1	knows the public pair (G_1, G_2) of isomorphic graphs, does not know Alice's secret π		
	Simulated Protocol		
2	randomly chooses a permutation ρ on $V(G_1)$ and a bit $a \in \{1, 2\}$, computes $H = \rho(G_a)$		
3		$\overset{H}{\Longrightarrow}$	
4			chooses a bit $b \in \{1, 2\}$ at random and wants to see an isomorphism between G_b and H
5		$\overset{b}{\Longleftarrow}$	
6	if $b \neq a$ then M deletes all messages transmitted in this round and repeats; if $b = a$ then M sends $\sigma = \rho$		
7		$\overset{\sigma}{\Longrightarrow}$	
8			$b = a$ implies that indeed $$\sigma(G_b) = H,$$ so Bob accepts "Alice's" identity

Fig. 14. How to simulate the Goldreich-Micali-Wigderson protocol without knowing the secret π.

It remains to show that the protocol in Figure 13 is zero-knowledge. Figure 14 shows a simulated protocol with Malice, who does not know the secret π, replacing Alice. The information communicated in one round of the protocol is given by a triple of the form (H, b, σ). Whenever Malice chooses a bit a with $a = b$, she simply sends $\sigma = \rho$ and wins: Bob, or any independent observer, will not detect that she in fact is Malice. Otherwise, whenever $a \neq b$, Malice fails. However, that's no problem at all: She simply deletes this round from the simulated protocol and repeats. Thus, she can produce a sequence of triples of the form (H, b, σ) that is

indistinguishable from the corresponding sequence of triples in the original protocol between Alice and Bob. It follows that the Goldreich-Micali-Wigderson protocol is zero-knowledge. ∎

4.4 Fiat and Shamir's Zero-Knowledge Protocol

Based on a similar protocol by Goldwasser, Micali and Rackoff [GMR89], Amos Fiat and Adi Shamir [FS86] proposed a zero-knowledge protocol for a number-theoretical problem. It is based on the assumption that computing square roots in $\mathbb{Z}_n^*$ is infeasible in practice. Due to its properties, the Fiat-Shamir protocol is particularly suitable for authentication of individuals in large computer networks. It is a public-key protocol, it is more efficient than other public-key protocols such as the RSA algorithm, it can be implemented on a chip card, and it is zero-knowledge. These advantages resulted in a rapid deployment of the protocol in practical applications. The Fiat-Shamir protocol is integrated in the "Videocrypt" Pay-TV system [CH91]. The original Fiat-Shamir identification scheme has later been improved by Feige, Fiat und Shamir [FFS88] to a zero-knowledge protocol in which not only the secret square roots modulo n are not revealed, but also the information of whether or not there *exists* a square root modulo n is not leaked.

The theory of zero-knowledge may also become important in future internet technologies. To prevent confusion, we note that Zero-Knowledge Systems, Inc., a Montréal-based company that was founded in 1997 and provides products and services enabling users to protect their privacy on-line on the world wide web, is not a commercial fielding of zero-knowledge protocols [Gol01a].

THEOREM 4.6. [FS86]   *The Fiat-Shamir procedure given in Figure 15 is a zero-knowledge protocol.*

PROOF. Look at Figure 15. The protocol is correct, since Alice knows the secret $s \in \mathbb{Z}_n^*$ that she has chosen, and thus she can compute $y = r \cdot s^b$, where b is the bit that Bob has chosen at random. Hence, it holds in $\mathbb{Z}_n^*$ that

$$y^2 \equiv (r \cdot s^b)^2 \equiv r^2 \cdot s^{2b} \equiv r^2 \cdot v^b \equiv x \cdot v^b \mod n,$$

so Bob accepts Alice's identity.

Suppose now that Erich or Malice want to cheat by pretending to be Alice. They do not know her secret s, nor do they know the primes p and q, but they do know the public $n = pq$ and $v = s^2 \mod n$. They want to convince Bob that they know Alice's secret s, the square root of v modulo n. If, by coincidence, Bob's bit b equals zero then $y = r \cdot s^0 = r$ and they win. However, if $b = 1$, computing a y that satisfies $y^2 \equiv x \cdot v^b \mod n$ requires knowledge of the secret s, assuming that computing square roots modulo n is hard. Without knowing s, if Malice or Erich were able to compute the correct answer for both $b = 0$ and $b = 1$, say y_b with $y_b^2 \equiv x \cdot v^b \mod n$, they could efficiently compute square roots modulo n as follows: $y_0^2 \equiv x \mod n$ and $y_1^2 \equiv x \cdot v \mod n$ implies $(\frac{y_1}{y_0})^2 \equiv v \mod n$; hence, $\frac{y_1}{y_0}$ is a square root of v modulo n.

It follows that they can cheat with probability at most $\frac{1}{2}$. Of course, they can always guess the bit b in advance and prepare the answer accordingly. Choosing $x = r^2 \cdot v^{-b} \mod n$ and $y = r$ implies that

$$y^2 \equiv r^2 \equiv r^2 \cdot v^{-b} \cdot v^b \equiv x \cdot v^b \mod n. \tag{4.16}$$

Step	Alice		Bob
Key generation			
1	chooses two large primes p and q and a secret $s \in \mathbb{Z}_n^*$, $n = pq$, and computes $v = s^2 \mod n$; p, q, and s are kept secret, whereas n and v are public		
Protocol			
2	chooses $r \in \mathbb{Z}_n^*$ at random and computes $x = r^2 \mod n$		
3		$\overset{x}{\Longrightarrow}$	
4			chooses a bit $b \in \{0, 1\}$ at random
5		$\overset{b}{\Longleftarrow}$	
6	computes $y = r \cdot s^b \mod n$		
7		$\overset{y}{\Longrightarrow}$	
8			verifies that indeed $$y^2 \equiv x \cdot v^b \mod n$$ and accepts accordingly

Fig. 15.   The Fiat-Shamir zero-knowledge protocol.

Thus, Bob will not detect any irregularities and will accept. Hence, their chance to cheat successfully is exactly $\frac{1}{2}$. Again, if Bob demands, say, k independent rounds of the protocol to be executed, he can make the cheating probability as small as desired and is very likely to detect any cheater.

It remains to show that the Fiat-Shamir protocol in Figure 15 is zero-knowledge. Figure 16 shows a simulated protocol with Malice, who does not know the secret s, replacing Alice. The information communicated in one round of the protocol is given by a triple of the form (x, b, y). In addition to the randomly chosen $r \in \mathbb{Z}_n^*$, Malice guesses a bit $c \in \{0, 1\}$ and computes $x = r^2 \cdot v^{-c} \mod n$, which she sends to Bob. Whenever c happens to be equal to Bob's bit b, Malice simply sends $y = r$ and wins. By an argument analogous to Equation (4.16) above, neither Bob nor any independent observer will detect that she actually is Malice:

$$y^2 \equiv r^2 \equiv r^2 \cdot v^{-c} \cdot v^b \equiv x \cdot v^b \mod n.$$

Otherwise, whenever $c \neq b$, Malice fails. However, that's no problem at all: She simply deletes this round from the simulated protocol and repeats. Thus, she can produce a sequence of triples of the form (x, b, y) that is indistinguishable from the corresponding sequence of triples in the original protocol between Alice and Bob. It follows that the Fiat-Shamir protocol is zero-knowledge. ∎

We have chosen to give here the original Fiat-Shamir identification scheme as

Step	Malice		Bob
	Simulated key generation		
1	knows the public $n = pq$ and $v = s^2 \mod n$; does not know the private primes p and q and Alice's secret s		
	Simulated Protocol		
2	randomly chooses $r \in \mathbb{Z}_n^*$ and a bit $c \in \{0, 1\}$, computes $x = r^2 \cdot v^{-c} \mod n$		
3		$\xRightarrow{x}$	
4			chooses a bit $b \in \{0, 1\}$ at random
5		$\xLeftarrow{b}$	
6	if $b \neq c$ then M deletes all messages transmitted in this round and repeats; if $b = c$ then M sends $y = r$		
7		$\xRightarrow{y}$	
8			$b = c$ implies that indeed $$y^2 = r^2 = r^2 v^{-c} v^b$$ $$\equiv x \cdot v^b \mod n,$$ so Bob accepts "Alice's" identity

Fig. 16.   How to simulate the Fiat-Shamir protocol without knowing the secret s.

presented in most books (see, e.g., [Gol01b,BSW01]). Note, however, that quite a number of modifications and improvements of the Fiat-Shamir protocol have been proposed, including the "zero-knowledge proof of knowledge" protocol of Feige, Fiat und Shamir [FFS88]. We also note in passing that we omitted many formal details in our arguments in this section. A rigid formalism (see [Gol01b]) is helpful in discussing many subtleties that can arise in zero-knowledge protocols. For example, looking at Figure 15, Alice could be impersonated by anyone who picks the value $r = 0$ without Bob detecting this fraud. We refer to Burmester and Desmedt [BD89] for appropriate modifications of the scheme. Moreover, Burmester et al. [BDPW89, BDB92] proposed efficient zero-knowledge protocols in a general algebraic setting.

5.   STRONGLY NONINVERTIBLE ASSOCIATIVE ONE-WAY FUNCTIONS

Recall Rivest and Sherman's secret-key agreement protocol (Figure 10) and Rabi and Sherman's digital signature protocol (Figure 11) presented in Section 3.4. Both of these protocols use a candidate for a strongly noninvertible, associative one-way

function. Are these protocols secure? This question has two aspects: (1) Are they secure under the assumption that strongly noninvertible, associative one-way functions indeed exist? (2) What evidence do we have for the existence of such functions?

The first question is an open problem. Security here depends on precisely how "strong noninvertibility" is defined, and in which model. Traditional complexity theory is concerned with the worst-case model and has identified a large number of problems that are hard in the worst case. Cryptographic applications, however, require the more demanding average-case model (see, e.g., [Gol01b,Gol99,Lub96]) for which much less is known. As noted by Rabi and Sherman [RS97], no proof of security for the Rivest–Sherman and Rabi–Sherman protocols is currently known, and even assuming the existence of associative one-way functions that are strongly noninvertible in the weaker worst-case model would not imply that the protocols are secure. In that regard, however, the Rivest–Sherman and Rabi–Sherman protocols are just like many other protocols currently used in practical applications. For example, neither the Diffie–Hellman protocol nor the RSA protocol currently has a proof of security. There are merely heuristic, intuitive arguments about how to avoid certain direct attacks. The "security" of the Diffie–Hellman protocol draws on the assumption that computing discrete logarithms is hard, and the "security" of the RSA protocol draws on the assumption that factoring large integers is hard. Breaking Diffie–Hellman is not even known to be as hard as the discrete logarithm problem, and breaking RSA is not even known to be as hard as the factoring problem. In a similar vein, Rabi and Sherman [RS93,RS97] only give intuitive arguments for the security of their protocols, explaining how to employ the strong noninvertibility of associative one-way functions to preclude certain direct attacks.

Turning to the second question raised above: What evidence do we have that strongly noninvertible, associative one-way functions exist? Assuming P $\neq$ NP, we will show how to construct total, strongly noninvertible, commutative,[6] associative one-way functions [HR99]. The question of whether or not P equals NP is perhaps the most important question in theoretical computer science. It is widely believed that P differs from NP, although this question has remained open for more than thirty years now. For more background on complexity theory, we refer to the textbooks [BDG95,BC93,HO02,Pap94].

5.1   Definitions and Progress of Results

From now on, we adopt the worst-case notion of one-way functions that is due to Grollmann and Selman [GS88], see also the papers by Ko [Ko85], Berman [Ber77], and Allender [All85,All86], and the surveys [Sel92,BHHR99]. Recall that one-way functions are easy to compute but hard to invert. To prevent the notion of non-invertibility from being trivialized, one-way functions are required to be "honest," i.e., to not shrink their inputs too much. Formal definitions of various types of honesty can be found in [GS88,HRW97,HR00,RH02,HPR01,Hom00,HT02].

One-way functions are often considered to be one-argument functions. Since the protocols from Section 3.4 require two-argument functions, the original definition

---

[6]Commutativity is needed to extend the Rivest–Sherman and Rabi–Sherman protocols from two parties to $m > 2$ parties.

is here tailored to the case of two-ary functions. Let $\rho : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ be any two-ary function; ρ may be nontotal and it may be many-to-one. We say that ρ is *(polynomial-time) invertible* if there exists a polynomial-time computable function g such that for all $z \in \text{image}(\rho)$, it holds that $\rho(g(z)) = z$; otherwise, we call ρ *not polynomial-time invertible*, or *noninvertible* for short. We say that ρ is a *one-way function* if and only if ρ is honest, polynomial-time computable, and noninvertible. One-argument one-way functions are well-known to exist if and only if $\text{P} \neq \text{NP}$; see, e.g., [Sel92,BDG95]. It is easy to prove the analogous result for two-argument one-way functions, see [HR99,RS97].

We now define strong noninvertibility (strongness, for short). As with noninvertibility, strongness requires an appropriate notion of honesty so as to not be trivial. This notion is called "s-honesty" in [HPR01], and since it is merely a technical requirement, we omit a formal definition here. Intuitively, "s-honesty" fits the notion of strong noninvertibility in that it is measured not only in the length of the function value but also in the length of the corresponding given argument.

Definition 5.1. (see [RS97,HR99])   Let $\sigma : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ be any two-ary function; σ may be nontotal and it may be many-to-one. Let $\langle \cdot, \cdot \rangle : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ be some standard pairing function.

(1) We say that σ is *(polynomial-time) invertible with respect to its first argument* if and only if there exists a polynomial-time computable function g_1 such that for all $z \in \text{image}(\sigma)$ and for all a and b with $(a,b) \in \text{domain}(\sigma)$ and $\sigma(a,b) = z$, it holds that $\sigma(a, g_1(\langle a, z \rangle)) = z$.

(2) We say that σ is *(polynomial-time) invertible with respect to its second argument* if and only if there exists a polynomial-time computable funtion g_2 such that for all $z \in \text{image}(\sigma)$ and for all a and b with $(a,b) \in \text{domain}(\sigma)$ and $\sigma(a,b) = z$, it holds that $\sigma(g_2(\langle b, z \rangle), b) = z$.

(3) We say that σ is *strongly noninvertible* if and only if σ is neither invertible with respect to its first argument nor invertible with respect to its second argument.

(4) We say that σ is a *strong one-way function* if and only if σ is s-honest, polynomial-time computable, and strongly noninvertible.

Below, we define Rabi and Sherman's notion of associativity, which henceforth will be called "weak associativity."

Definition 5.2. [RS93,RS97]   A two-ary function $\sigma : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ is said to be *weakly associative* if and only if $\sigma(a, \sigma(b,c)) = \sigma(\sigma(a,b),c)$ holds for all $a,b,c \in \mathbb{N}$ for which each of (a,b), (b,c), $(a,\sigma(b,c))$, and $(\sigma(a,b),c)$ belongs to the domain of σ.

Although this notion is suitable for total functions, weak associativity does not adequately fit the nontotal function case. More precisely, weak associativity fails to preclude, for nontotal functions, equations from having a defined value to the left, while being undefined to the right of their equality sign. Therefore, we present in Definition 5.3 below another notion of associativity for two-ary functions that is suitable both for total and for nontotal two-ary functions. This definition is due to Hemaspaandra and Rothe [HR99] who note that the two notions of associativity are provably distinct (see Proposition 5.4), and this distinction can be explained

(see [HR99]) via Kleene's careful discussion [Kle52, pp. 327–328] of two distinct notions of equality for partial functions in recursion theory: "Weak equality" between two partial functions explicitly allows "specific, defined function values being equal to undefined" as long as the functions take the same values on their joint domain. In contrast, "complete equality" precludes this unnatural behavior by additionally requiring that two given partial functions be equal only if their domains coincide; i.e., whenever one is undefined, so is the other. Weak associativity from Definition 5.2 is based on Kleene's weak equality between partial functions, whereas associativity from Definition 5.3 is based on Kleene's complete equality.

Definition 5.3. [HR99]   Let $\sigma : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ be any two-ary function; σ may be nontotal. Define $\mathbb{N}_\perp = \mathbb{N} \cup \{\perp\}$, and define an extension $\overset{\perp}{\sigma} \colon \mathbb{N}_\perp \times \mathbb{N}_\perp \to \mathbb{N}_\perp$ of σ as follows:

$$\overset{\perp}{\sigma}(a,b) = \begin{cases} \sigma(a,b) & \text{if } a \neq \perp \text{ and } b \neq \perp \text{ and } (a,b) \in \text{domain}(\sigma) \\ \perp & \text{otherwise.} \end{cases}$$

We say that σ is *associative* if and only if, for all $a, b, c \in \mathbb{N}$, it holds that

$$\overset{\perp}{\sigma}(\overset{\perp}{\sigma}(a,b),c) = \overset{\perp}{\sigma}(a,\overset{\perp}{\sigma}(b,c)).$$

We say that σ is *commutative* if and only if, for all $a, b \in \mathbb{N}$, it holds that

$$\overset{\perp}{\sigma}(a,b) = \overset{\perp}{\sigma}(b,a).$$

The following proposition explores the relation between the two associativity notions presented respectively in Definition 5.2 and in Definition 5.3. In particular, these are indeed different notions.

PROPOSITION 5.4. [HR99]

(1) *Every associative two-ary function is weakly associative.*

(2) *Every total two-ary function is associative exactly if it is weakly associative.*

(3) *There exist two-ary functions that are weakly associative, yet not associative.*

Rabi and Sherman [RS93,RS97] showed that P $\neq$ NP if and only if commutative, weakly associative one-way functions exist. However, they did not achieve strong noninvertibility. They did not achieve totality of their weakly associative one-way functions, although they presented a construction that they claimed achieves totality of any weakly associative one-way function. Hemaspaandra and Rothe [HR99] showed that Rabi and Sherman's claim is unlikely to be true: Any proof of this claim would imply that NP = UP, which is considered to be unlikely. Intuitively, the reason that Rabi and Sherman's construction is unlikely to work is that the functions constructed in [RS93,RS97] are not associative in the sense of Definition 5.3. In contrast, the Rabi–Sherman construction indeed is useful to achieve totality of the associative, strongly noninvertible one-way functions constructed in [HR99].

Thus, Rabi and Sherman [RS93,RS97] left open the question of whether there are plausible complexity-theoretic conditions sufficient to ensure the existence of total, strongly noninvertible, commutative, associative one-way functions. They also asked whether such functions could be *constructed* from any given one-way function. Section 5.2 presents the answers to these questions.

5.2 Creating Strongly Noninvertible, Total, Commutative, Associative One-Way Functions from Any One-Way Function

Theorem 5.5 below is the main result of this section. Since $P \neq NP$ is equivalent to the existence of one-way functions with no additional properties required, the converse of the implication stated in Theorem 5.5 is clearly also true. However, we focus on only the interesting implication directions in Theorem 5.5 and in the upcoming Theorem 5.7 and Theorem 5.9.

THEOREM 5.5. [HR99]   *If* $P \neq NP$ *then there exist total, strongly noninvertible, commutative, associative one-way functions.*

A detailed proof of Theorem 5.5 can be found in [HR99], see also the survey [BHHR99]. Here, we briefly sketch the proof idea.

Assume $P \neq NP$. Let A be a set in $NP - P$, and let M be a fixed NP machine accepting A. Let $x \in A$ be an input accepted by M in time $p(|x|)$, where p is some polynomial. A useful property of NP sets is that they have polynomial-time checkable certificates.[7] That is, for each certificate z for "$x \in A$," it holds that: (a) the length of z is polynomially bounded in the length of x, and (b) z certifies membership of x in A in a way that can be verified deterministically in polynomial time. $\texttt{Certificates}_M(x)$ denotes the set of all certificates of M on input x. Note that $\texttt{Certificates}_M(x)$ is nonempty exactly if $x \in A$.
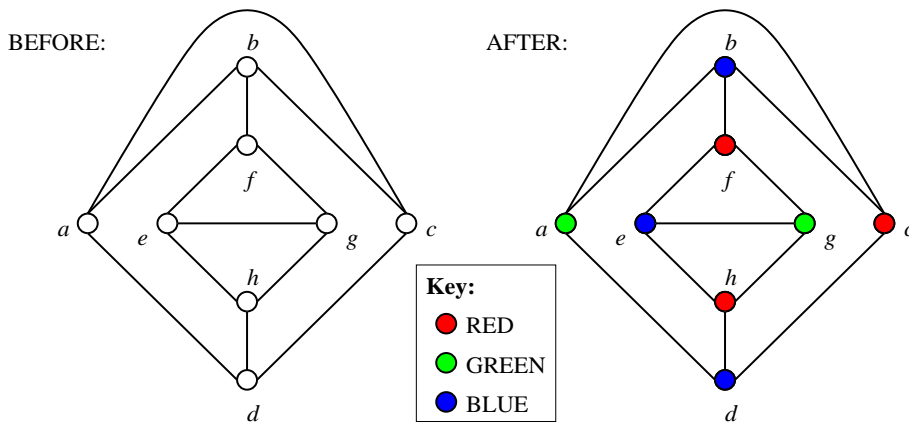

Fig. 17.   The three-coloring ψ of graph G.

Example 5.6. For concreteness, consider `Graph-Three-Colorability`, a well-known NP-complete problem that asks whether the vertices of a given graph can be colored with three colors such that no two adjacent vertices receive the same color. Such a coloring is called a legal three-coloring. In other words, a legal three-coloring is a mapping ψ from the vertex set of G to the set of colors (RED, GREEN,

---

[7]Other common names for "certificate" are "witness" and "proof" and "solution."

BLUE) such that the resulting color classes are independent sets. Figure 17 gives an example.

The standard NP machine for `Graph-Three-Colorability` works as follows: Given a graph G, nondeterministically guess a three-coloring ψ of G (i.e., a partition of the vertex set of G into three color classes) and check deterministically whether ψ is legal.

Any legal three-coloring of G is a *certificate* for the three-colorability of G (with respect to the above NP machine). For the specific graph from Figure 17, one certificate ψ is specified by the three color classes $\psi^{-1}(\text{GREEN}) = \{a, g\}$, $\psi^{-1}(\text{RED}) = \{c, f, h\}$, and $\psi^{-1}(\text{BLUE}) = \{b, d, e\}$.

As is standard, graphs as well as three-colorings can be encoded as binary strings that represent nonnegative integers.

Suppose that for each $x \in A$ and for each certificate z for "$x \in A$," it holds that $|z| = p(|x|) > |x|$. This is only a technical requirement that makes it easy to tell input strings apart from their certificates. For any integers $u, v, w \in \mathbb{N}$, let $\min(u, v)$ denote the minimum of u and v, and let $\min(u, v, w)$ denote the minimum of u, v, and w. Define a two-ary function $\sigma : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ as follows:

—If $a = \langle x, z_1 \rangle$ and $b = \langle x, z_2 \rangle$ for some $x \in A$ with certificates $z_1, z_2 \in \texttt{Certificates}_M(x)$ (where, possibly, $z_1 = z_2$), then define $\sigma(a, b) = \langle x, \min(z_1, z_2) \rangle$;

—if there exists some $x \in A$ with certificate $z \in \texttt{Certificates}_M(x)$ such that either $a = \langle x, x \rangle$ and $b = \langle x, z \rangle$, or $a = \langle x, z \rangle$ and $b = \langle x, x \rangle$, then define $\sigma(a, b) = \langle x, x \rangle$;

—otherwise, $\sigma(a, b)$ is undefined.

What is the intuition behind the definition of σ? The number of certificates contained in the arguments of σ is decreased by one in a way that ensures the associativity of σ. Moreover, σ is noninvertible, and it is also strongly noninvertible. Why? The intuition here is that, regardless of whether none or either one of its arguments is given in addition to σ's function value, the inversion of σ requires information about the certificates for elements of A. However, our assumption that $A \notin \text{P}$ guarantees that this information cannot efficiently be extracted.

One can show that σ is a commutative, associative one-way function that is strongly noninvertible. We will show associativity and strongness below. Note that σ is not a total function. However, σ can be extended to a total function without losing any of its other properties already established [HR99].

We now show that σ is strongly noninvertible. For a contradiction, suppose there is a polynomial-time computable inverter, g_2, for a fixed second argument. Hence, for each $w \in \text{image}(\sigma)$ and for each second argument b for which there is an $a \in \mathbb{N}$ with $\sigma(a, b) = w$, it holds that

$$\sigma(g_2(\langle b, w \rangle), b) = w.$$

Then, contradicting our assumption that $A \notin \text{P}$, one could decide A in polynomial time as follows:

On input x, compute $g_2(\langle \langle x, x \rangle, \langle x, x \rangle \rangle)$, compute the integers d and e for which $\langle d, e \rangle$ equals $g_2(\langle \langle x, x \rangle, \langle x, x \rangle \rangle)$, and accept x if and only if $d = x$ and $e \in \texttt{Certificates}_M(x)$.

Hence, σ is not invertible with respect to its second argument. An analogous argument shows that σ is not invertible with respect to its first argument. Thus, σ is strongly noninvertible.

Next, we prove that σ is associative. Let $\overset{\perp}{\sigma}$ be the total extension of σ as in Definition 5.3. Fix any three elements of $\mathbb{N}$, say $a = \langle a_1, a_2 \rangle$, $b = \langle b_1, b_2 \rangle$, and $c = \langle c_1, c_2 \rangle$. To show that

$$\overset{\perp}{\sigma}(\overset{\perp}{\sigma}(a,b),c) \;=\; \overset{\perp}{\sigma}(a,\overset{\perp}{\sigma}(b,c)) \tag{5.17}$$

holds, distinguish two cases.

Case 1:. $a_1 = b_1 = c_1$ and $\{a_2, b_2, c_2\} \subseteq \{a_1\} \cup \mathtt{Certificates}_M(a_1)$.

Let $x, y \in \{a, b, c\}$ be any two fixed arguments of σ. As noted above, if x and y together contain i certificates for "$a_1 \in A$," where $i \in \{1, 2\}$, then $\sigma(x, y)$—and thus also $\overset{\perp}{\sigma}(x, y)$—contains exactly $\max\{0, i-1\}$ certificates for "$a_1 \in A$." In particular, $\overset{\perp}{\sigma}(x, y)$ preserves the minimum certificate if both x and y contain a certificate for "$a_1 \in A$."

If exactly one of x and y contains a certificate for "$a_1 \in A$," then $\overset{\perp}{\sigma}(x, y) = \langle a_1, a_1 \rangle$.

If none of x and y contains a certificate for "$a_1 \in A$," then $\sigma(x, y)$ is undefined, so $\overset{\perp}{\sigma}(x, y) = \perp$.

Let $k \leq 3$ be a number telling us how many of a_2, b_2, and c_2 belong to $\mathtt{Certificates}_M(a_1)$. For example, if $a_2 = b_2 = c_2 \in \mathtt{Certificates}_M(a_1)$ then $k = 3$. Consequently:

—If $k \leq 1$ then both $\overset{\perp}{\sigma}(\overset{\perp}{\sigma}(a,b),c)$ and $\overset{\perp}{\sigma}(a,\overset{\perp}{\sigma}(b,c))$ equals $\perp$.

—If $k = 2$ then both $\overset{\perp}{\sigma}(\overset{\perp}{\sigma}(a,b),c)$ and $\overset{\perp}{\sigma}(a,\overset{\perp}{\sigma}(b,c))$ equals $\langle a_1, a_1 \rangle$.

—If $k = 3$ then both $\overset{\perp}{\sigma}(\overset{\perp}{\sigma}(a,b),c)$ and $\overset{\perp}{\sigma}(a,\overset{\perp}{\sigma}(b,c))$ equals $\langle a_1, \min(a_2, b_2, c_2) \rangle$.

In each of these three cases, Equation (5.17) is satisfied.

Case 2:. Suppose Case 1 is not true.

Then, either it holds that $a_1 \neq b_1$ or $a_1 \neq c_1$ or $b_1 \neq c_1$, or it holds that $a_1 = b_1 = c_1$ and $\{a_2, b_2, c_2\}$ is not contained in $\{a_1\} \cup \mathtt{Certificates}_M(a_1)$. By the definition of σ, in both cases it follows that

$$\overset{\perp}{\sigma}(\overset{\perp}{\sigma}(a,b),c) \;=\; \perp \;=\; \overset{\perp}{\sigma}(a,\overset{\perp}{\sigma}(b,c)),$$

which satisfies Equation (5.17) and concludes the proof that σ is associative.

Finally, we mention some related results of Chris Homan [Hom00] who studied upper and lower bounds on the ambiguity of associative one-way functions. In particular, extending Rabi and Sherman's [RS97] result that no total, associative one-way function is injective, he proved that no total, associative one-way function can be constant-to-one. He also showed that, under the plausible assumption that $P \neq UP$, there exist linear-to-one, total, strongly noninvertible, associative one-way functions.

On a slightly less related note, Homan and Thakur [HT02] recently proved that one-way permutations (i.e., one-way functions that are total, one-to-one, and onto)

exist if and only if P $\neq$ UP $\cap$ coUP. This result gives a characterization of one-way permutations in terms of a complexity class separation, and thus the ultimate answer to a question studied in [GS88,HRW97,HR00,RH02].

5.3   If P $\neq$ NP then Some Strongly Noninvertible Functions are Invertible

Is every strongly noninvertible function noninvertible? Hemaspaandra, Pasanen, and Rothe [HPR01] obtained the surprising result that if P $\neq$ NP then this is not necessarily the case. This result shows that the term "strong noninvertibility" introduced in [RS93,RS97] actually is a misnomer, since it seems to suggest that strong noninvertibility always implies noninvertibility, which is not true.

THEOREM 5.7. [HPR01]   *If* P $\neq$ NP *then there exists a total, honest two-ary function that is strongly one-way but not a one-way function.*

We give a brief sketch of the proof. Assume P $\neq$ NP. Then, there exists a total two-ary one-way function, call it ρ. For any integer $n \in \mathbb{N}$, define the notation

$$\text{odd}(n) = 2n + 1 \quad \text{and} \quad \text{even}(n) = 2n.$$

Define a function $\sigma : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ as follows. Let $a, b \in \mathbb{N}$ be any two arguments of σ.

—If $a \neq 0 \neq b$, $a = \langle x, y \rangle$ is odd, and b is even, then define $\sigma(a, b) = \text{even}(\rho(x, y))$.
—If $a \neq 0 \neq b$, a is even, and $b = \langle x, y \rangle$ is odd, then define $\sigma(a, b) = \text{even}(\rho(x, y))$.
—If $a \neq 0 \neq b$, and a is odd if and only if b is odd, then define $\sigma(a, b) = \text{odd}(a + b)$.
—If $a = 0$ or $b = 0$, then define $\sigma(a, b) = a + b$.

We claim that σ is strongly noninvertible. For a contradiction, suppose σ were invertible with respect to its first argument via an inverter, g_1. By the definition of σ, for any $z \in \text{image}(\rho)$ with $z \neq 0$, the function g_1 on input $\langle 2, \text{even}(z) \rangle$ yields an odd integer b from which we can read the pair $\langle x, y \rangle$ with $\rho(x, y) = z$. Hence, using g_1, one could invert ρ in polynomial time, a contradiction. Thus, σ is not invertible with respect to its first argument. Analogously, one can show that σ is not invertible with respect to its second argument. So, σ indeed is strongly noninvertible.

But σ is invertible! By the fourth line in the definition of σ, every z in the image of σ has a preimage of the form $(0, z)$. Thus, the function g defined by $g(z) = (0, z)$ inverts σ in polynomial time. Hence, σ is not a one-way function.

Why don't we use a different notion of strongness that automatically implies noninvertibility? Here is an attempt to redefine the notion of strongness accordingly, which yields a new notion that we will call "overstrongness."

Definition 5.8. [HPR01]   Let $\sigma : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ be any two-ary function; σ may be nontotal and it may be many-to-one. We say that σ is *overstrong* if and only if no polynomial-time computable function f with $f : \{1, 2\} \times \mathbb{N} \times \mathbb{N} \to \mathbb{N} \times \mathbb{N}$ satisfies that for each $i \in \{1, 2\}$ and for each $z, a \in \mathbb{N}$:

$$((\exists b \in \mathbb{N})[(\sigma(a, b) = z \wedge i = 1) \vee (\sigma(b, a) = z \wedge i = 2)]) \implies \sigma(f(i, z, a)) = z.$$

Note that overstrongness implies both noninvertibility and strong noninvertibility. However, the problem with this new definition is that it completely loses the

core of why strongness precludes direct attacks on the Rivest–Sherman and Rabi–Sherman protocols protocols. To see why, look at Figure 10 and Figure 11, which give the protocols of Rabi, Rivest, and Sherman. In contrast to overstrongness, Rabi, Rivest, and Sherman's original definition of strong noninvertibility (see Definition 5.1) *respects the argument given.* It is this feature that precludes Erich from being able to compute Alice's secret x from the transmitted values $\sigma(x, y)$ and y, which he knows. In short, overstrongness is *not well-motivated* by the protocols of Rabi, Rivest, and Sherman.

We mention without proof some further results of Hemaspaandra, Pasanen, and Rothe [HPR01].

THEOREM 5.9. [HPR01]

(*1*) *If* $\mathrm{P} \neq \mathrm{NP}$ *then there exists a total, honest, s-honest, two-ary overstrong function. Consequently, if* $\mathrm{P} \neq \mathrm{NP}$ *then there exists a total two-ary function that is both one-way and strongly one-way.*

(*2*) *If* $\mathrm{P} \neq \mathrm{NP}$ *then there exists a total, s-honest two-ary one-way function* σ *such that* σ *is invertible with respect to its first argument and* σ *is invertible with respect to its second argument.*

(*3*) *If* $\mathrm{P} \neq \mathrm{NP}$ *then there exists a total, s-honest two-ary one-way function that is invertible with respect to either one of its arguments (thus, it is not strongly one-way), yet that is not invertible with respect to its other argument.*

(*4*) *If* $\mathrm{P} \neq \mathrm{NP}$ *then there exists a total, honest, s-honest two-ary function that is noninvertible and strongly noninvertible but that is not overstrong.*

ACKNOWLEDGMENTS

REFERENCES

M. Ajtai and C. Dwork. A public-key cryptosystem with worst-case/average-case equivalence. In *Proceedings of the 29th ACM Symposium on Theory of Computing*, pages 284–293. ACM Press, 1997.

M. Ajtai. Generating hard instances of lattice problems. In *Proceedings of the 28th ACM Symposium on Theory of Computing*, pages 99–108. ACM Press, 1996.

M. Ajtai.   The shortest vector problem in L_2 is NP-hard for randomized reductions.   In *Proceedings of the 30th ACM Symposium on Theory of Computing*, pages 10–19.   ACM Press, 1998.   Full version available on-line as ECCC TR97-047 at ftp://ftp.eccc.uni-trier.de/pub/eccc/reports/1997/TR97-047/index.html.

M. Agrawal, N. Kayal, and N. Saxena. PRIMES is in P. Unpublished manuscript, August 2002.

E. Allender. Invertible functions, 1985. PhD thesis, Georgia Institute of Technology.

E. Allender. The complexity of sparse sets in P. In *Proceedings of the 1st Structure in Complexity Theory Conference*, pages 1–11. Springer-Verlag *Lecture Notes in Computer Science #223*, June 1986.

L. Babai. Trading group theory for randomness. In *Proceedings of the 17th ACM Symposium on Theory of Computing*, pages 421–429. ACM Press, April 1985.

F. Bauer. *Decrypted Secrets: Methods and Maxims of Cryptology*. Springer-Verlag, second edition, 2000.

Gilles Brassard and Claude Crepeau. Sorting out zero-knowledge. In *Advances in Cryptology—EUROCRYPT 89*, pages 181–191. Springer-Verlag *Lecture Notes in Computer Science #434*, April 1989.

D. Bovet and P. Crescenzi. *Introduction to the Theory of Complexity*. Prentice Hall, 1993.

M. Burmester and Y. Desmedt. Remarks on the soundness of proofs. *Electronics Letters*, 25:1509–1511, 1989.

D. Boneh and G. Durfee. Cryptanalysis of RSA with private key d less than $N^{0.292}$. *IEEE Transactions on Information Theory*, IT-46, 2000.

M. Burmester, Y. Desmedt, and T. Beth. Efficient zero-knowledge identification schemes for smart cards. *The Computer Journal*, 35(1):21–29, February 1992.

J. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity II*. EATCS Monographs on Theoretical Computer Science. Springer-Verlag, 1990.

J. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity I*. EATCS Monographs on Theoretical Computer Science. Springer-Verlag, second edition, 1995.

M. Burmester, Y. Desmedt, F. Piper, and M. Walker. A general zero-knowledge scheme. In *Advances in Cryptology—EUROCRYPT 89*, pages 122–133. Springer-Verlag *Lecture Notes in Computer Science #434*, April 1989.

L. Berman. *Polynomial Reducibilities and Complete Sets*. PhD thesis, Cornell University, Ithaca, NY, 1977.

A. Beutelspacher. *Cryptology*. Spectrum series. Mathematical Association of America, 1994.

A. Beygelzimer, L. Hemaspaandra, C. Homan, and J. Rothe. One-way functions in worst-case cryptography: Algebraic and security properties are on the house. *SIGACT News*, 30(4):25–40, December 1999.

L. Babai and S. Moran. Arthur-Merlin games: A randomized proof system, and a hierarchy of complexity classes. *Journal of Computer and System Sciences*, 36(2):254–276, 1988.

D. Boneh. Twenty years of attacks on the RSA cryptosystem. *Notices of the AMS*, 46(2):203–213, February 1999.

A. Beutelspacher, J. Schwenk, and K. Wolfenstetter. *Moderne Verfahren der Kryptographie*. Vieweg, 4th edition, 2001. In German.

J. Buchmann. *Introduction to Cryptography*. Undergraduate Texts in Mathematics. Springer-Verlag, 2001.

J. Cai. Some recent progress on the complexity of lattice problems. In *Proceedings of the 14th Annual IEEE Conference on Computational Complexity*, pages 158–179. IEEE Computer Society Press, May 1999.

M. Cohen and J. Hashkes. A system for controlling access to broadcast transmissions. European Patent Application 0 428252 A2, May 1991.

D. Coppersmith. Small solutions to polynomial equations, and low exponent RSA vulnerabilities. *Journal of Cryptology*, 10(4):233–260, 1997.

W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.

T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, IT-31(4):469–472, 1985.

J. Feigenbaum. Overview of interactive proof systems and zero-knowledge. In G. Simmons, editor, *Contemporary Cryptology: The Science of Information Integrity*, pages 423–439. IEEE Computer Society Press, 1992.

U. Feige, A. Fiat, and A. Shamir. Zero-knowledge proofs of identity. *Journal of Cryptology*, 1(2):77–94, 1988.

A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology – CRYPTO '86*, pages 186–194. Springer-Verlag *Lecture Notes in Computer Science #263*, 1986.

J. Gill. Computational complexity of probabilistic Turing machines. *SIAM Journal on Computing*, 6(4):675–695, 1977.

S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. In *Proceedings of the 17th ACM Symposium on Theory of Computing*, pages 291–304. ACM Press, April 1985.

S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, February 1989.

O. Goldreich, Y. Mansour, and M. Sipser. Interactive proof systems: Provers that never fail and random selection. In *Proceedings of the 28th IEEE Symposium on Foundations of Computer Science*, pages 449–461. IEEE Computer Society Press, October 1987.

O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity and a methodology of cryptographic protocol design. In *Proceedings of the 27th IEEE Symposium on Foundations of Computer Science*, pages 174–187, April 1986.

O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *Journal of the ACM*, 38(3):691–729, July 1991.

O. Goldreich. Randomness, interactive proofs, and zero-knowledge—A survey. In R. Herken, editor, *The Universal Turing Machine: A Half-Century Survey*, pages 377–405. Oxford University Press, Oxford, 1988.

S. Goldwasser. Interactive proof systems. In J. Hartmanis, editor, *Computational Complexity Theory*, pages 108–128. AMS Short Course Lecture Notes: Introductory Survey Lectures, Proceedings of Symposia in Applied Mathematics, Volume 38, American Mathematical Society, 1989.

O. Goldreich. A taxonomy of proof systems. In L. Hemaspaandra and A. Selman, editors, *Complexity Theory Retrospective II*, pages 109–134. Springer-Verlag, 1997.

O. Goldreich. *Modern cryptography, probabilistic proofs, and pseudorandomness*, volume 17 of *Algorithms and Combinatorics*. Springer-Verlag, 1999.

I. Goldberg, November 13, 2001. Personal Communication.

O. Goldreich. *Foundations of Cryptography*. Cambridge University Press, 2001.

J. Grollmann and A. Selman. Complexity measures for public-key cryptosystems. *SIAM Journal on Computing*, 17(2):309–335, 1988.

S. Goldwasser and M. Sipser. Private coins versus public coins in interactive proof systems. In S. Micali, editor, *Randomness and Computation*, volume 5 of *Advances in Computing Research*, pages 73–90. JAI Press, Greenwich, 1989. A preliminary version appeared in *Proc. 18th Ann. ACM Symp. on Theory of Computing*, 1986, pp. 59–68.

J. Håstad. Solving simultaneous modular equations of low degree. *SIAM Journal on Computing*, 17(2):336–341, April 1988. Special issue on cryptography.

L. Hemaspaandra and M. Ogihara. *The Complexity Theory Companion*. EATCS Texts in Theoretical Computer Science. Springer-Verlag, 2002.

C. Homan. Low ambiguity in strong, total, associative, one-way functions. Technical Report TR-734, University of Rochester, Department of Computer Science, Rochester, NY, August 2000.

L. Hemaspaandra, K. Pasanen, and J. Rothe. If P ≠ NP then some strongly noninvertible functions are invertible. In *Proceedings of the 13th International Symposium on Fundamentals of Computation Theory*, pages 162–171. Springer-Verlag *Lecture Notes in Computer Science #2138*, August 2001.

L. Hemaspaandra and J. Rothe. Creating strong, total, commutative, associative one-way functions from any one-way function in complexity theory. *Journal of Computer and System Sciences*, 58(3):648–659, June 1999.

L. Hemaspaandra and J. Rothe. Characterizing the existence of one-way permutations. *Theoretical Computer Science*, 244(1–2):257–261, August 2000.

L. Hemaspaandra, J. Rothe, and G. Wechsung. On sets with easy certificates and the existence of one-way permutations. In *Proceedings of the Third Italian Conference on Algorithms and Complexity*, pages 264–275. Springer-Verlag *Lecture Notes in Computer Science #1203*, March 1997.

C. Homan and M. Thakur. One-way permutations and self-witnessing languages. In *Proceedings of the 2nd IFIP International Conference on Theoretical Computer Science, Stream 1 of the 17th IFIP World Computer Congress*, pages 243–254. Kluwer Academic Publishers, August 2002. To appear.

G. Hardy and E. Wright. *An Introduction to the Theory of Numbers*. Clarendon Press, Oxford, 5th edition, 1979.

D. Kahn. *The Codebreakers: The Story of Secret Writing*. MacMillan Publishing Company, 1967.

S. Kleene. *Introduction to Metamathematics*. D. van Nostrand Company, Inc., New York and Toronto, 1952.

D. Knuth. *The Art of Computer Programming: Seminumerical Algorithms*, volume 2 of *Computer Science and Information*. Addison-Wesley, second edition, 1981.

K. Ko. On some natural complete operators. *Theoretical Computer Science*, 37(1):1–30, 1985.

B. Kaliski Jr. and M. Robshaw. The secure use of RSA. *CryptoBytes*, 1(3):7–13, 1995.

R. Kumar and D. Sivakumar. Complexity of SVP—A reader's digest. *SIGACT News*, 32(3):40–52, June 2001.

J. Köbler, U. Schöning, and J. Torán. Graph isomorphism is low for PP. *Computational Complexity*, 2:301–330, 1992.

J. Köbler, U. Schöning, and J. Torán. *The Graph Isomorphism Problem: Its Structural Complexity*. Birkhäuser, 1993.

H. Lenstra Jr. Factoring integers with elliptic curves. *Annals of Mathematics*, 126:649–673, 1987.

A. Lenstra and H. Lenstra, Jr. *The Development of the Number Field Sieve*, volume 1554 of *Lecture Notes in Mathematics*. Springer-Verlag, 1993.

M. Luby. *Pseudorandomness and Cryptographic Applications*. Princeton Computer Science Notes. Princeton University Press, Princeton, New Jersey, 1996.

D. Micciancio. The shortest vector in a lattice is hard to approximate to within some constant. *SIAM Journal on Computing*, 30(6):2008–2035, March 2001. Preliminary version appeared in FOCS 1998.

G. Miller. Riemann's hypothesis and tests for primality. *Journal of Computer and System Sciences*, 13:300–317, 1976.

J. Moore. Protocol failures in cryptosystems. In G. Simmons, editor, *Contemporary Cryptology: The Science of Information Integrity*, pages 541–558. IEEE Computer Society Press, 1992.

U. Maurer and S. Wolf. The relationship between breaking the Diffie-Hellman protocol and computing discrete logarithms. *SIAM Journal on Computing*, 28(5):1689–1721, 1999.

National Institute of Standards and Technology (NIST). Digital signature standard (DSS). *Federal Register*, 56(169), August 1991.

National Institute of Standards and Technology (NIST). The Digital Signature Standard, proposed by NIST. *Communications of the Association for Computing Machinery*, 35(7):36–40, July 1992.

P. Nguyen and J. Stern. The two faces of lattices in cryptology. In *Proceedings of the International Conference on Cryptography and Lattices*, pages 146–180. Springer-Verlag *Lecture Notes in Computer Science #2146*, 2001.

C. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

J. Pollard. Theorems on factorization and primality testing. *Proc. Cambridge Philos. Soc.*, 76:521–528, 1974.

C. Pomerance and J. Sorenson. Counting the integers factorable via cyclotomic methods. *Journal of Algorithms*, 19(2):250–265, September 1995.

M. Rabin. Probabilistic algorithms for testing primality. *Journal of Number Theory*, 12:128–138, 1980.

J. Rothe and L. Hemaspaandra. On characterizing the existence of partial one-way permutations. *Information Processing Letters*, 82(3):165–171, May 2002.

M. Rabi and A. Sherman. Associative one-way functions: A new paradigm for secret-key agreement and digital signatures. Technical Report CS-TR-3183/UMIACS-TR-93-124, Department of Computer Science, University of Maryland, College Park, Maryland, 1993.

M. Rabi and A. Sherman. An observation on associative one-way functions in complexity theory. *Information Processing Letters*, 64(5):239–244, 1997.

R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signature and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

A. Salomaa. *Public-Key Cryptography*, volume 23 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, second edition, 1996.

U. Schöning. Graph isomorphism is in the low hierarchy. *Journal of Computer and System Sciences*, 37:312–323, 1987.

C. Schnorr. Efficient identification and signature schemes for smart cards. In *Advances in Cryptology – CRYPTO '89*, pages 239–251. Springer-Verlag *Lecture Notes in Computer Science #435*, February 1990.

B. Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. John Wiley and Sons, New York, second edition, 1996.

A. Selman. A survey of one-way functions in complexity theory. *Mathematical Systems Theory*, 25(3):203–221, 1992.

C. Shannon. Communication theory of secrecy systems. *Bell System Technical Journal*, 28(4):657–715, 1949.

A. Shamir. IP = PSPACE. *Journal of the ACM*, 39(4):869–877, 1992.

A. Shamir. RSA for paranoids. *CryptoBytes*, 1(3):1–4, 1995.

P. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, 1997.

G. Simmons. Symmetric and asymmetric encryption. *ACM Computing Surveys*, 11(4):305–330, 1979.

S. Singh. *The Code Book. The Science of Secrecy from Ancient Egypt to Quantum Cryptography*. Fourth Estate, London, 1999.

G. Simmons and M. Norris. Preliminary comments on the MIT public-key cryptosystem. *Cryptologia*, 1(4):406–414, 1977.

R. Solovay and V. Strassen. A fast Monte Carlo test for primality. *SIAM Journal on Computing*, 6:84–85, 1977. Erratum appears in the same journal, 7(1):118, 1978.

D. Stinson. *Cryptography Theory and Practice*. CRC Press, Boca Raton, 1995.

L. Valiant. The relative complexity of checking and evaluating. *Information Processing Letters*, 5(1):20–23, 1976.

D. Welsh. *Codes and Cryptography*. Oxford science publications. Clarendon Press, Oxford, 6th edition, 1998. Reprinted with corrections.

M. Wiener. Cryptanalysis of short RSA secret exponents. *IEEE Transactions on Information Theory*, IT-36(3):553–558, 1990.