CHAPMAN & HALL/CRC
CRYPTOGRAPHY AND NETWORK SECURITY

Jonathan Katz
Yehuda Lindell

Introduction to MODERN CRYPTOGRAPHY

Third Edition

CRC Press
Taylor & Francis Group
A CHAPMAN & HALL BOOK

Introduction to Modern Cryptography

Chapman & Hall/CRC Cryptography and Network Security Series

Introduction to Modern Cryptography Jonathan Katz and Yehuda Lindell

Series Editors: Douglas R. Stinson and Jonathan Katz

Secret History: The Story of Cryptology, Second Edition
Craig P. Bauer

Data Science for Mathematicians
Nathan Carter

Discrete Explorations
Craig P. Bauer

Cryptography: Theory and Practice, Fourth Edition
Douglas R. Stinson and Mary P. Paterson

Cryptology: Classical and Modern, Second Edition
Richard Klima and Neil Sigmon

Group Theoretic Cryptography
Maria Isabel Gonzalez Vasco and Rainer Steinwandt

Advances of DNA Computing in Cryptography
Suyel Namasudra and Ganesh Chandra Deka

Mathematical Foundations of Public Key Cryptography
Xiaoyun Wang, Guangwu Xu, Minggiang Wang, Xianmeng Meng

Guide to Pairing-Based Cryptography
Nadia El Mrabet and Marc Joye

https://www.crcpress.com/Chapman--HallCRC-Cryptography-and-Network-Security-Series/book-series/CHCRYNETSEC

Introduction to Modern Cryptography

Third Edition

Jonathan Katz and Yehuda Lindell

CRC Press
Taylor & Francis Group
Boca Raton London New York

CRC Press is an imprint of the
Taylor & Francis Group, an **informa** business
A CHAPMAN & HALL BOOK

Third edition published 2021
by CRC Press
6000 Broken Sound Parkway NW, Suite 300, Boca Raton, FL 33487-2742

and by CRC Press
2 Park Square, Milton Park, Abingdon, Oxon, OX14 4RN

First edition published by Taylor and Francis 2007 Second edition published by Taylor and Francis 2014

CRC Press is an imprint of Taylor & Francis Group, LLC

ISBN: 780815354369 (hbk)
ISBN: 9781351133036 (ebk)

Typeset in Computer Modern font
by KnowledgeWorks Global Ltd.

Visit the companion website/eResources:[insert CW/eResources URL

To Jill, Abigail, and Rena
– JK

To Yael, Yehonatan, Itamar, Orel, Shirel, and Noam
– YL

Contents

Preface

The goal of our book remains the same as in the first edition: to present the core paradigms and principles of modern cryptography to a general audience with a basic mathematics background. We have designed this book to serve as a textbook for undergraduate- or graduate-level courses in cryptography (in computer science, electrical engineering, or mathematics departments), as a general introduction suitable for self-study (especially for beginning graduate students), and as a reference for students, researchers, and practitioners.

There are numerous other cryptography textbooks available today, and the reader may rightly ask whether another book on the subject is needed. We would not have written this book—nor worked on revising it for the second and third editions—if the answer to that question were anything other than an unequivocal *yes*. What, in our opinion, distinguishes our book from others is that it provides a *rigorous* treatment of modern cryptography in an *accessible* and *introductory* manner.

Our focus is on *modern* (post-1980s) cryptography, which is distinguished from classical cryptography by its emphasis on definitions, precise assumptions, and rigorous proofs of security. We briefly discuss each of these in turn (these principles are explored in greater detail in Chapter 1):

- **The central role of definitions:** A key intellectual contribution of modern cryptography has been the recognition that *formal definitions of security are an essential first step in the design of any cryptographic primitive or protocol*. The reason, in retrospect, is simple: if you don't know what it is you are trying to achieve, how can you hope to know when you have achieved it? As we will see in this book, cryptographic definitions of security are quite strong and—at first glance—may appear impossible to achieve. One of the most amazing aspects of cryptography is that efficient constructions satisfying such strong definitions can be proven to exist (under rather mild assumptions).

- **The importance of precise assumptions:** As will be explained in Chapters 2 and 3, many cryptographic constructions cannot currently be proven secure unconditionally. Security, instead, generally relies on some widely believed (though unproven) assumption(s). The modern cryptographic approach dictates that *any such assumptions must be clearly stated and unambiguously defined*. This not only allows for objective evaluation of the assumptions but, more importantly, enables rigorous proofs of security (as described next).

- **The possibility of proofs of security:** The previous two principles serve as the basis for the idea that *cryptographic constructions can be proven secure* with respect to clearly stated definitions of security and relative to well-defined cryptographic assumptions. This concept is the essence of modern cryptography, and is what has transformed the field from an art to a science.

  The importance of this idea cannot be overemphasized. Historically, cryptographic schemes were designed in a largely heuristic fashion, and were deemed to be secure if the designers themselves could not find any attacks. In contrast, modern cryptography advocates the design of schemes with formal, mathematical proofs of security in well-defined models. Such schemes are *guaranteed* to be secure (with respect to a certain security definition) unless the underlying assumption is false. By relying on long-standing assumptions, it is thus possible to obtain schemes that are extremely unlikely to be broken.

A unified approach. The above principles of modern cryptography are relevant not only to the "theory of cryptography" community. The importance of precise definitions is, by now, widely understood and appreciated by developers and security engineers who use cryptographic tools to build secure systems, and rigorous proofs of security have become one of the requirements for cryptographic schemes to be standardized.

Changes in the Third Edition

In preparing the third edition, we have continued to integrate a more practical perspective without sacrificing a rigorous approach. This is reflected in a number of changes and additions as compared to the second edition:

- We have divided our treatment of symmetric-key encryption into two parts: Chapter 3 deals with security against "passive" attacks (i.e., CPA-security), while Chapter 5 addresses "active" attacks (i.e., CCA-security and authenticated encryption). Besides breaking up what was previously a long chapter, this also allows us to introduce message authentication codes before discussing active attacks against encryption schemes.

- With an eye toward symmetric-key schemes used in practice, we have improved our coverage of stream ciphers and stream-cipher modes of operation (Sections 3.6.1 and 3.6.2); added a treatment of nonce-based encryption (Section 3.6.4); and incorporated material about standardized schemes such as GMAC and Poly1305 (Section 4.5) as well as GCM, CCM, and ChaCha20-Poly1305 (Section 5.3.2).

- With similar motivation, we have added sections on the ChaCha20 stream cipher and SHA-3 to Chapter 7. As part of our discussion about SHA-3, we also describe the sponge construction.

- We have further increased our coverage of elliptic-curve cryptography (Section 9.3.4), including a discussion of elliptic curves used in practice.
- Our treatment of TLS in Section 13.7 has been updated to reflect the latest version (TLS 1.3).
- Reflecting recent trends, we have added a chapter (Chapter 14) describing the impact of quantum computers on cryptography, and providing examples of "post-quantum" encryption and signature schemes.

For those currently using the first edition of our book, as well as for reference, we also summarize the changes/additions we have already made in the second edition (all of which remain here):

- We have increased our coverage of *stream ciphers*, including stream-cipher modes of operation as well as stream-cipher design principles and examples of stream ciphers used in practice.
- We have emphasized the importance of authenticated encryption and secure communication sessions in Sections 5.2–5.4.
- We have moved our treatment of hash functions into its own chapter (Chapter 6), and have added a section on hash-function design principles and widely used constructions (Section 7.3). We have also improved our treatment of generic attacks on hash functions, including a discussion of rainbow tables (Section 6.4.3).
- We have included several important attacks on cryptographic *implementations* that arise in practice, including chosen-plaintext attacks on chained-CBC encryption (Section 3.6.3), timing attacks on MAC verification (Section 4.2), and padding-oracle attacks on CBC-mode encryption (Section 5.1.1).
- After much deliberation, we have decided to introduce the random-oracle model earlier in the book (Section 6.5). This has several benefits, including allowing for an integrated treatment of standardized public-key encryption and signature schemes in Chapters 12 and 13.
- We have strengthened our coverage of elliptic-curve cryptography (Section 9.3.4) and have added a discussion of its impact on recommended key lengths (Section 10.4).
- In the chapter on public-key encryption, we introduce the KEM/DEM paradigm as a form of hybrid encryption (see Section 12.3). We also cover DHIES/ECIES in addition to the RSA PKCS #1 standards.
- In the chapter on digital signatures, we now describe the construction of signatures from identification schemes using the Fiat–Shamir transform, with the Schnorr signature scheme as a prototypical example. We have

also improved our coverage of DSA/ECDSA. We include brief discussions of SSL/TLS and signcryption, both of which serve as culminations of material covered up to that point.

- In the "advanced topics" chapter, we have amplified our treatment of homomorphic encryption, and have added sections on secret sharing and threshold encryption.

Beyond the above, we have also edited the entire book to make extensive corrections as well as smaller adjustments, including more worked examples, to improve the exposition. Several additional exercises have also been added.

Guide to Using This Book

This section is intended primarily for instructors seeking to adopt this book for their course, though the student picking up this book on his or her own may also find it a useful overview.

Required background. We have structured the book so the only formal prerequisite is a course on discrete mathematics. Even here we rely on very little: we only assume familiarity with basic (discrete) probability and modular arithmetic. Students reading this book are also expected to have had some exposure to algorithms, mainly to be comfortable reading pseudocode and to be familiar with big-$\mathcal{O}$ notation. Many of these concepts are reviewed in Appendix A and/or when first used in the book.

Notwithstanding the above, the book does use definitions, proofs, and abstract mathematical concepts, and therefore requires some mathematical maturity. In particular, the reader is assumed to have had some exposure to proofs, whether in an upper-level mathematics course or a course on discrete mathematics, algorithms, or computability theory.

Suggestions for course organization. The core material of this book, which we recommend should be covered in any introductory course on cryptography, consists of the following (in all cases, starred sections are excluded; more on this below):

- *Introduction and Classical Cryptography:* Chapters 1 and 2 discuss classical cryptography and set the stage for modern cryptography.

- *Private-Key (Symmetric) Cryptography:* Chapter 3–5 provide a thorough treatment of private-key encryption and message authentication, and Chapter 6 covers hash functions and their applications. (Section 6.6 could be skipped if that material will not be used later.)

  We also highly recommend covering at least part of Chapter 7, which deals with symmetric-key primitives used in practice; in our experience students really enjoy this material, and it makes the abstract ideas they

have learned in previous chapters more concrete. Although we do consider this core material, it is not used in the remainder of the book and so can be safely skipped if desired.

- *Public-Key Cryptography:* Chapter 9 gives a self-contained introduction to all the number theory needed for the remainder of the book. The material in The public-key revolution, including Diffie–Hellman key exchange, is described in Chapter 11. Chapters 12 and 13 go into detail about public-key encryption and digital signatures; those pressed for time can pick and choose what to cover appropriately.

We are typically able to cover most of the above in a one-semester (35-hour) undergraduate or Masters-level course (omitting some proofs and skipping some topics, as needed) or, with some changes to add more material on theoretical foundations, in the first three-quarters of a one-semester PhD-level course. Instructors with more time available can proceed at a more leisurely pace or incorporate additional topics, as discussed below.

Those wishing to cover additional material, in either a longer course or a faster-paced graduate course, will find that the book is structured to allow flexible incorporation of other topics as time permits (and depending on the interests of the instructor). Specifically, the starred (*) sections and chapters may be covered in any order, or skipped entirely, without affecting the overall flow of the book. We have taken care to ensure that none of the core (i.e., unstarred) material depends on any of the starred material and, for the most part, the starred sections do not depend on each other. (When they do, this dependence is explicitly noted.)

We suggest the following from among the starred topics for those wishing to give their course a particular flavor:

- *Theory:* A more theoretically inclined course could include material from Section 3.2.2 (semantic security); Chapter 8 (one-way functions and hard-core predicates, and constructing pseudorandom generators, functions, and permutations from one-way permutations); Section 9.4 (one-way functions and collision-resistant hash functions from number-theoretic assumptions); Section 12.5.3 (RSA encryption without random oracles); and Section 15.3 (cryptographic protocols).

- *Mathematics:* A course directed at students with a strong mathematics background—or being taught by someone who enjoys this aspect of cryptography—could incorporate Section 4.6 (information-theoretic MACs in finite fields); some of the more advanced number theory from Chapter 9 (e.g., the Chinese remainder theorem, the Miller–Rabin primality test, and more on elliptic curves); and all of Chapter 10 (algorithms for factoring and computing discrete logarithms).

In either case, a selection of advanced public-key schemes from Chapters 14 and 15 could also be included.

Feedback and Errata

Our goal in writing this book was to make modern cryptography accessible to a wide audience beyond the "theoretical computer science" community. We hope you will let us know if we have succeeded! The many enthusiastic emails we have received in response to our first and second editions have made the whole process of writing this book worthwhile.

We are always happy to receive feedback. We hope there are no errors or typos in the book; if you do find any, however, we would greatly appreciate it if you let us know. You can email your comments and errata to `jkatz2@gmail.com` and `lindell@biu.ac.il`; please put "Introduction to Modern Cryptography" in the subject line. A list of known errata will be maintained at `http://www.cs.umd.edu/~jkatz/imc.html`.

Acknowledgments

We continue to be grateful to all those who have sent us comments, suggestions, and corrections for the book. We would like to thank, in particular, Jack Aaron, Rounak Agarwal, Ionut Ambrosie, Dan Bernstein, Jeremiah Blocki, David Cash, Claude Crépeau, Dana Dachman-Soled, Daniel Escudero, Pooya Farshim, Rolf Haenni, Imededdine Jerbi, Ali El Kaafarani, Zach Kissel, Angelique Faye Loe, Wilde Luo, Tal Malkin, Alejandro Mardones, Kurt Pan, Greg Plaxton, Kyle Andrew Porter, Christian Schaffner, Jim Tallent, Hanh Tang, Markus Triska, and Rui Xue for their feedback on the second edition.

Finally, we thank our wives and children for all their support during the now over a decade(!) we have spent working on this project.

Part I

Introduction and Classical Cryptography

Chapter 1

Introduction

1.1 Cryptography and Modern Cryptography

The *Concise Oxford English Dictionary (9th ed.)* defines cryptography as "the art of writing or solving codes." This is historically accurate, but does not capture the current breadth of the field or its modern scientific foundations. The definition focuses solely on the *codes* that have been used for centuries to enable secret communication. But cryptography nowadays encompasses much more than this: it deals with mechanisms for ensuring integrity, techniques for exchanging secret keys, protocols for authenticating users, electronic voting, cryptocurrency, and more. Without attempting to provide a complete characterization, we would say that modern cryptography involves *the study of mathematical techniques for securing digital information, systems, and distributed computations against adversarial attacks.*

The dictionary definition also refers to cryptography as an *art*. Until late in the 20th century cryptography was, indeed, largely an art. Constructing good codes, or breaking existing ones, relied on creativity and a developed sense of how codes work. There was little theory to rely on and, for a long time, no working definition of what constitutes a good code. Beginning in the 1970s and 1980s, this picture of cryptography radically changed. A rich theory began to emerge, enabling the rigorous study of cryptography as a *science* and a mathematical discipline. This perspective has, in turn, influenced how researchers think about the broader field of computer security.

Another very important difference between classical cryptography (say, before the 1980s) and modern cryptography relates to its adoption. Historically, the major consumers of cryptography were military organizations and governments. Today, cryptography is everywhere! If you have ever authenticated yourself by typing a password, purchased something by credit card over the Internet, or downloaded a verified update for your operating system, you have used cryptography. And, more and more, programmers with relatively little experience are being asked to "secure" the applications they write by incorporating cryptographic mechanisms.

In short, cryptography has gone from a heuristic set of techniques for ensuring secret communication for a few niche applications to a science that helps secure systems more generally for ordinary people around the world.

Goals of this book. Our goal is to make the basic principles of modern cryptography accessible to students of computer science, electrical engineering, or mathematics; to professionals who want to incorporate cryptography in systems or software they are developing; and to anyone with a basic level of mathematical maturity who is interested in understanding this fascinating field. After completing this book, the reader should appreciate the security guarantees common cryptographic primitives are intended to provide; be aware of standard (secure) constructions of such primitives; and be able to perform a basic evaluation of new schemes based on their proofs of security (or lack thereof) and the mathematical assumptions underlying those proofs. It is not our intention for readers to become experts—or to be able to design new cryptosystems—after finishing this book, but we have attempted to provide the terminology and foundational material needed for the interested reader to subsequently study the more advanced literature in this field.

This chapter. The focus of this book is the formal study of *modern* cryptography, but we begin in this chapter with a more informal discussion of "classical" cryptography. Besides allowing us to ease into the material, our treatment in this chapter will also serve to motivate the more rigorous approach we will be taking in the rest of the book. Our intention here is not to be exhaustive and, as such, this chapter should not be taken as a representative historical account. The reader interested in the history of cryptography is invited to consult the references at the end of this chapter.

1.2 The Setting of Private-Key Encryption

Classical cryptography was concerned with designing and using *codes* (or *ciphers*) that enable two parties to send messages while keeping those messages hidden from an eavesdropper who can monitor all communication between them. In modern parlance, codes are called *encryption schemes* and that is the terminology we will use here. Security of all classical encryption schemes relies on a secret—a *key*—shared by the communicating parties in advance and unknown to the eavesdropper. This scenario, in which the communicating parties share some secret information in advance, is known as the *private-key* (or *shared-/secret-key*) setting, and *private-key encryption* is one example of a cryptographic primitive used in this setting. Before describing some historical encryption schemes, we discuss private-key encryption more generally.

In the context of private-key encryption, two parties share a key and use that key when they want to communicate secretly. One party can send a message, or *plaintext*, to the other by using the shared key to *encrypt* (or "scramble") the message and thus obtain a *ciphertext* that is transmitted to the receiver. The receiver uses the same key to *decrypt* (or "unscramble") the

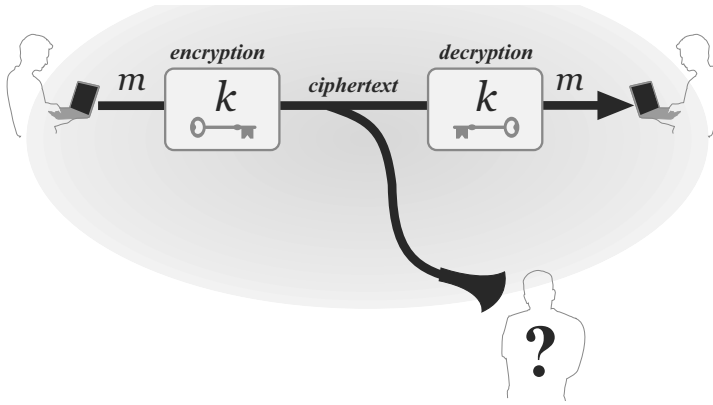

FIGURE 1.1: One common use case for private-key cryptography (here, encryption): two parties share a key that they use to communicate securely.

ciphertext and recover the original message. The same key is used to convert the plaintext into a ciphertext and back; that is why this setting is also known as the *symmetric-key* setting, where the symmetry lies in the fact that both parties hold the same key that is used for encryption and decryption. This is in contrast to *asymmetric*, or *public-key*, encryption (introduced in Chapter 11), where encryption and decryption use different keys.

As already noted, the goal of encryption is to keep the plaintext hidden from an eavesdropper who can monitor the communication channel and observe the ciphertext. We discuss this in more detail later in this chapter, and spend a great deal of time in Chapters 2, 3, and 5 formally defining this goal.

There are two canonical applications of private-key cryptography. In the first (cf. Figure 1.1), the two communication parties are separated in *space*, e.g., a worker in New York communicating with her colleague in California. These two users are assumed to have been able to securely share a key in advance of their communication. (Note that if one party simply sends the key to the other over the public communication channel, then the eavesdropper obtains the key also!) This could be accomplished, for example, by having the parties physically meet in a secure location to share a key before they separate; in the example just given, the co-workers might arrange to share a key when they are both in the New York office. In other cases, sharing a key securely is more difficult. For the next several chapters we simply assume that sharing a key is possible; we revisit this issue in Chapter 11.

The second widespread application of private-key cryptography involves the same party communicating with itself over *time*. (See Figure 1.2.) Consider, e.g., disk encryption, where a user encrypts some plaintext and stores the resulting ciphertext on his hard drive; the same user will return at a later point in time to decrypt the ciphertext and recover the original data. The hard drive here serves as the communication channel on which an attacker

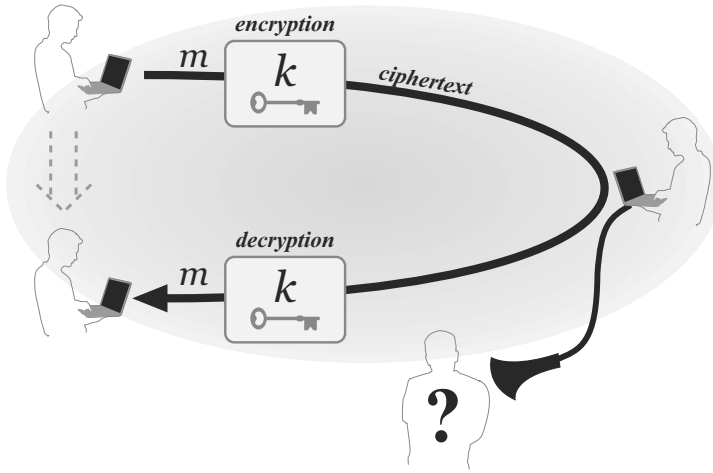

FIGURE 1.2: Another common use case of private-key cryptography (again, encryption): a single user stores data securely over time.

might eavesdrop if it can gain access to the hard drive and read its contents. "Sharing" the key is now trivial, though the user still needs a secure and reliable way to remember/store the key for use at a later point in time.

The syntax of encryption. Formally, a private-key encryption scheme is defined by specifying a *message space* $\mathcal{M}$ along with three algorithms: a procedure for generating keys (Gen), a procedure for encrypting (Enc), and a procedure for decrypting (Dec). The message space $\mathcal{M}$ defines the set of "legal" messages, i.e., those supported by the scheme. The algorithms of the scheme have the following functionality:

1. The *key-generation algorithm* Gen is a probabilistic algorithm that outputs a key k chosen according to some distribution.
2. The *encryption algorithm* Enc takes as input a key k and a message m and outputs a ciphertext c. We denote by $\mathsf{Enc}_k(m)$ the encryption of the plaintext m using the key k.
3. The *decryption algorithm* Dec takes as input a key k and a ciphertext c and outputs a plaintext m. We denote the decryption of the ciphertext c using the key k by $\mathsf{Dec}_k(c)$.

An encryption scheme must satisfy the following correctness requirement: for every key k output by Gen and every message $m \in \mathcal{M}$, it holds that

$$\mathsf{Dec}_k(\mathsf{Enc}_k(m)) = m.$$

In words: encrypting a message and then decrypting the resulting ciphertext using the same key yields the original message.

The set of all possible keys output by the key-generation algorithm is called the *key space* and is denoted by $\mathcal{K}$. Almost always, Gen simply chooses a key uniformly from the key space; in fact, one can assume without loss of generality that this is the case (see Exercise 2.1).

Reviewing our earlier discussion, an encryption scheme can be used by two parties who wish to communicate secretly as follows. First, Gen is run to obtain a key k that the parties share. Later, when one party wants to send a plaintext m to the other, she computes $c := \mathsf{Enc}_k(m)$ and sends the resulting ciphertext c over the public channel to the other party.[1] Upon receiving c, the other party computes $m := \mathsf{Dec}_k(c)$ to recover the original plaintext.

Keys and Kerckhoffs' principle. As should be clear from the above, if an eavesdropping adversary knows the algorithm Dec as well as the key k shared by the two communicating parties, then that adversary will be able to decrypt any ciphertexts transmitted by those parties. It is for this reason that the communicating parties must share the key k securely and keep k completely hidden from everyone else. Perhaps they should keep the decryption algorithm Dec secret, also? For that matter, would it not be better for them to keep all the details of the encryption scheme secret?

Auguste Kerckhoffs [114, 115] argued the opposite in the late 19th century when elucidating several design principles for military ciphers. One of the most important of these, now known simply as *Kerckhoffs' principle*, was:

> *The cipher method must not be required to be secret, and it must be able to fall into the hands of the enemy without inconvenience.*

That is, an encryption scheme should be designed to be secure *even if* an eavesdropper knows all the details of the scheme, so long as the attacker doesn't know the key being used. Stated differently, security should not rely on the encryption scheme being secret; instead, Kerckhoffs' principle demands that *security rely solely on secrecy of the key.*

There are three primary arguments in favor of Kerckhoffs' principle. The first is that it is significantly easier to maintain secrecy of a short key than to keep secret a (more complicated) encryption scheme. This is especially true when encryption is used widely. For example, consider the case where encryption is used for communication between all pairs of employees in some organization. Unless each pair of parties use their own, unique scheme, some parties will know the scheme being used by others. Moreover, information about the scheme might be leaked by one of those employees (say, after being fired), or obtained by an attacker using reverse engineering. In short, it is simply unrealistic to assume that the encryption scheme will remain secret.

Second, in case the honest parties' shared, secret information *is* ever exposed, it will be much easier for them to change the key than to replace the

[1] We use ":=" to denote deterministic assignment, and assume for now that Enc is deterministic. A list of common notation can be found in the back of the book.

encryption scheme. (Consider updating a file versus installing a new program.) Moreover, it is relatively trivial to generate a new random secret, whereas it would be a huge undertaking to design a new encryption scheme.

Finally, prior to widespread deployment of an encryption scheme, there is a significant benefit to encouraging public review of that scheme in order to check for possible weaknesses. Going further, it is desirable to *standardize* encryption schemes so that (1) compatibility between different users is ensured and (2) the general public will use strong schemes that have undergone public scrutiny. Overall, perhaps counter-intuitively, it is advantageous to have broad, public dissemination of the full details of an encryption scheme—the exact opposite of keeping the scheme secret.

Nowadays Kerckhoffs' principle is understood as advocating that the entire cryptographic design process be made completely public, in stark contrast to the notion of "security by obscurity" that suggests keeping algorithms secret improves security. In fact, it is very dangerous to use a proprietary, "home-brewed" algorithm (i.e., a non-standardized algorithm designed in secret) since published designs undergo public peer review and are therefore likely to be stronger. Many years of experience have demonstrated that it is very difficult to construct good cryptographic schemes. Our confidence in the security of a scheme is much higher if it has been extensively studied by experts (beyond the designers of the scheme) and no flaws have been found. As simple and obvious as it may sound, the principle of open cryptographic design (i.e., Kerckhoffs' principle) has been ignored over and over again with disastrous results. Fortunately, today there are several secure, standardized, and widely available cryptosystems and no reason to use anything else.

1.3 Historical Ciphers and Their Cryptanalysis

In our study of "classical" cryptography we will examine some historical encryption schemes and show that they are insecure. Our main aims in presenting this material are (1) to highlight the weaknesses of heuristic approaches to cryptography, and thus motivate the modern, rigorous approach that will be taken in the rest of the book, and (2) to demonstrate that simple approaches to achieving secure encryption are unlikely to succeed. Along the way, we will present some central principles of cryptography inspired by the weaknesses of these historical schemes.

In this section, plaintext characters are written in `lower case` and ciphertext characters are written in `UPPER CASE` for clarity.

Caesar's cipher. One of the oldest recorded ciphers, known as *Caesar's cipher*, is described in *De Vita Caesarum, Divus Iulius* ("The Lives of the Caesars, the Deified Julius"), written in approximately 110 CE:

> *There are also letters of his to Cicero, as well as to his intimates on private affairs, and in the latter, if he had anything confidential to say, he wrote it in cipher, that is, by so changing the order of the letters of the alphabet, that not a word could be made out...*

Julius Caesar encrypted by shifting the letters of the alphabet 3 places forward: `a` was replaced with `D`, `b` with `E`, and so on. At the very end of the alphabet, the letters wrap around and so `z` was replaced with `C`, `y` with `B`, and `x` with `A`. For example, encryption of the message `begin the attack now`, with spaces removed, gives:

`EHJLQWKHDWWDFNQRZ`.

An immediate problem with this cipher is that the encryption method is *fixed*; there is no key. Thus, anyone learning how Caesar encrypted his messages would be able to decrypt effortlessly.

Interestingly, a variant of this cipher called ROT-13 (where the shift is 13 places instead of 3) is still used nowadays in various online forums. It is understood that this does not provide any cryptographic security; it is used merely to ensure that the text (say, a movie spoiler) is unintelligible unless the reader of a message makes the conscious decision to decrypt it.

The shift cipher and the sufficient key-space principle. The *shift cipher* can be viewed as a keyed variant of Caesar's cipher.[2] Specifically, in the shift cipher the key k is a number between 0 and 25. To encrypt, letters are shifted as in Caesar's cipher, but now by k places. Mapping this to the syntax of encryption described earlier, the message space consists of arbitrary length strings of English letters with punctuation, spaces, and numerals removed, and with no distinction between upper and lower case. Algorithm Gen outputs a uniform key $k \in \{0, \ldots, 25\}$; algorithm Enc takes a key k and a plaintext and shifts each letter of the plaintext forward k positions (wrapping around at the end of the alphabet); and algorithm Dec takes a key k and a ciphertext and shifts every letter of the ciphertext *backward* k positions.

A more mathematical description is obtained by equating the English alphabet with the set $\{0, \ldots, 25\}$ (so $\mathtt{a} = 0$, $\mathtt{b} = 1$, etc.). The message space $\mathcal{M}$ is then any finite sequence of integers from this set. Encryption of the message $m = m_1 \cdots m_\ell$ (where $m_i \in \{0, \ldots, 25\}$) using key k is given by

$$\mathsf{Enc}_k(m_1 \cdots m_\ell) = c_1 \cdots c_\ell, \quad \text{where } c_i = [(m_i + k) \bmod 26].$$

(The notation $[a \bmod N]$ denotes the remainder of a upon division by N, with $0 \leq [a \bmod N] < N$. We refer to the process mapping a to $[a \bmod N]$ as *reduction modulo* N; see also Chapter 9.) Decryption of a ciphertext $c = c_1 \cdots c_\ell$ using key k is given by

$$\mathsf{Dec}_k(c_1 \cdots c_\ell) = m_1 \cdots m_\ell, \quad \text{where } m_i = [(c_i - k) \bmod 26].$$

[2] In some books, "Caesar's cipher" and "shift cipher" are used interchangeably.

Is the shift cipher secure? Before reading on, try to decrypt the following ciphertext that was generated using the shift cipher and a secret key k:

OVDTHUFWVZZPISLRLFZHYLAOLYL.

Is it possible to recover the message without knowing k? Actually, it is trivial! The reason is that there are only 26 possible keys. So one can try to decrypt the ciphertext using every possible key and thereby obtain a list of 26 candidate plaintexts. The correct plaintext will certainly be on this list; moreover, if the ciphertext is "long enough" then the correct plaintext will likely be the only candidate on the list that "makes sense." By scanning the list of candidates it is easy to recover the original plaintext. (This is not necessarily true, but will be true most of the time. Even when it is not, this attack narrows down the set of potential plaintexts to at most 26 possibilities.)

An attack that involves trying every possible key is called a *brute-force* or *exhaustive-search* attack. Clearly, for an encryption scheme to be secure it must not be vulnerable to such an attack.[3] This observation is known as the *sufficient key-space principle*:

> *Any secure encryption scheme must have a key space that is sufficiently large to make an exhaustive-search attack infeasible.*

One can debate what amount of effort makes a task "infeasible," and an exact determination of feasibility depends on both the resources of a potential attacker and the length of time for which the sender and receiver want to ensure secrecy of their communication. Nowadays, attackers can use supercomputers, thousands of cloud servers, or graphics processing units (GPUs) to speed up brute-force attacks. To protect against such attacks the key space must be very large—say, of size at least 2^{80}, and even larger in many settings.

The sufficient key-space principle gives a *necessary* condition for security, but not a *sufficient* one. The next example demonstrates this.

The mono-alphabetic substitution cipher. In the shift cipher, the key defines a map from each letter of the (plaintext) alphabet to some letter of the (ciphertext) alphabet, where the map is a fixed shift determined by the key. In the *mono-alphabetic substitution cipher* the key also defines a map on the alphabet, but the map is now allowed to be *arbitrary* subject only to the constraint that it be one-to-one (so that decryption is possible). The key space thus consists of all *bijections*, or *permutations*, of the alphabet. So, for example, the key that defines the following permutation

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
X	E	U	A	D	N	B	K	V	M	R	O	C	Q	F	S	Y	H	W	G	L	Z	I	J	P	T

[3] Technically, this is only true if the message space is larger than the key space; we will return to this point in Chapter 2. Encryption schemes used in practice have this property.

(in which `a` maps to `X`, etc.) would encrypt the message `tellhimaboutme` to `GDOOKVCXEFLGCD`. The name of this cipher comes from the fact that the key defines a (fixed) substitution for individual characters of the plaintext.

Assuming the English alphabet is being used, the key space is of size $26! = 26 \cdot 25 \cdot 24 \cdots 2 \cdot 1$, or approximately 2^{88}, and a brute-force attack is infeasible. This, however, does not mean the cipher is secure! In fact, as we will show next, it is easy to break this scheme even though it has a large key space.

Assume English-language text is being encrypted (i.e., the text is grammatically correct English writing, not just text written using characters of the English alphabet). The mono-alphabetic substitution cipher can then be attacked by utilizing statistical properties of the English language. (Of course, the same idea works for any language.) The attack relies on the facts that:

1. For any key, the mapping of each letter is fixed, and so if `e` is mapped to `D`, then every appearance of `e` in the plaintext will result in the appearance of `D` in the ciphertext.

2. The frequency distribution of individual letters in English-language text is known. (See Figure 1.3.) Of course, very short texts may deviate from this distribution, but even texts consisting of only a few sentences tend to have distributions that are very close to it.

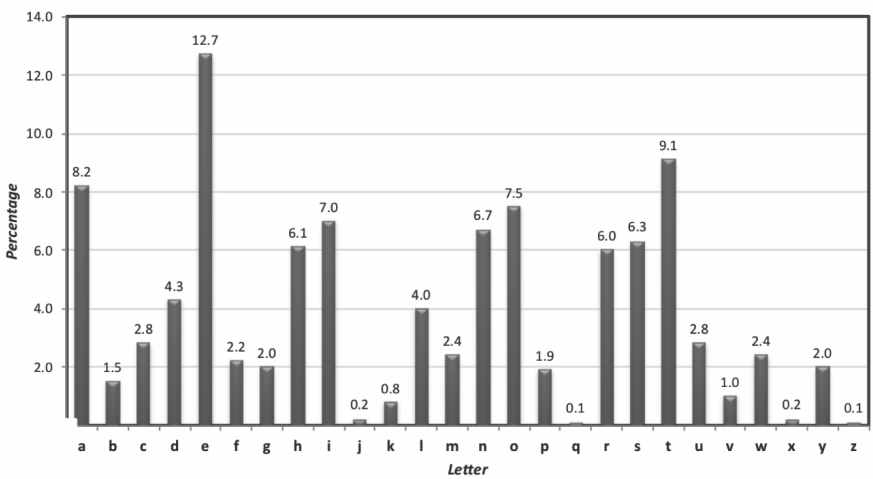

FIGURE 1.3: Average letter frequencies for English-language text.

The attack works by tabulating the frequency distribution of characters in the ciphertext, i.e., recording that `A` appeared 12% of the time, `B` appeared 3% of the time, and so on. These frequencies are then compared to the known letter frequencies of normal English text. One can then guess parts of the mapping defined by the key based on the observed frequencies. For example,

since `e` is the most frequent letter in English, one can guess that the most frequent character in the ciphertext corresponds to the plaintext character `e`, and so on. Some of the guesses may be wrong, but enough of the guesses will be correct to enable relatively quick decryption (especially utilizing other knowledge of English, such as the fact that `q` is generally followed by `u`, and that `h` is likely to appear between `t` and `e`). We conclude that although the mono-alphabetic substitution cipher has a large key space, it is still insecure.

It should not be surprising that the mono-alphabetic substitution cipher can be quickly broken, since puzzles based on this cipher are common (and are solved by some people before their morning coffee!). We recommend that you try to decipher the following ciphertext—this should convince you how easy it is to carry out the attack. (Use Figure 1.3.)

```
JGRMQOYGHMVBJWRWQFPWHGFFDQGFPFZRKBEEBJIZQQOCIBZKLFAFGQVFZFWWE
OGWOPFGFHWOLPHLRLOLFDMFGQWBLWBWQOLKFWBYLBLYLFSFLJGRMQBOLWJVFP
FWQVHQWFFPQOQVFPQOCFPOGFWFJIGFQVHLHLROQVFGWJVFPFOLFHGQVQVFILE
OGQILHQFQGIQVVOSFAFGBWQVHQWIJVWJVFPFWHGFIWIHZZRQGBABHZQOCGFHX
```

An improved attack on the shift cipher. We can use letter-frequency tables to give an improved attack on the shift cipher. Our previous attack on the shift cipher required decrypting the ciphertext using each possible key, and then checking which key results in a plaintext that "makes sense." A drawback of this approach is that it is somewhat difficult to automate, since it is difficult for a computer to check whether a given plaintext "makes sense." (We do not claim that it would be impossible, as the attack could be automated using a dictionary of valid English words.) More importantly, there may be cases—we will see one later—where the plaintext characters follow the same distribution as English-language text even though the plaintext itself is not valid English, in which case checking for a plaintext that "makes sense" will not work.

We now describe an attack that does not suffer from these drawbacks. As before, associate the letters of the English alphabet with $0, \ldots, 25$. Let p_i, with $0 \leq p_i \leq 1$, denote the frequency of the ith letter in normal English text (i.e., $p_0 = 0.082$ using Figure 1.3). Calculation using Figure 1.3 gives

$$\sum_{i=0}^{25} p_i^2 \approx 0.065. \tag{1.1}$$

Now, say we are given some ciphertext and let q_i denote the frequency of the ith letter of the alphabet in this ciphertext; i.e., q_i is simply the number of occurrences of the ith letter of the alphabet in the ciphertext divided by the length of the ciphertext. If the key is k, then p_i should be roughly equal to q_{i+k} for all i because the ith letter is mapped to the $(i+k)$th letter. (We use $i+k$ instead of the more cumbersome $[i+k \bmod 26]$.) Thus, if we compute

$$I_j \stackrel{\text{def}}{=} \sum_{i=0}^{25} p_i \cdot q_{i+j} \tag{1.2}$$

for each value of $j \in \{0, \ldots, 25\}$, then we expect to find that $I_k \approx 0.065$ (where k is the actual key), whereas I_j for $j \neq k$ will be different from 0.065. This leads to a key-recovery attack that is easy to automate: compute I_j for all j, and then output the value k for which I_k is closest to 0.065.

The Vigenère (poly-alphabetic shift) cipher. The statistical attack on the mono-alphabetic substitution cipher can be carried out because the key defines a fixed mapping that is applied letter-by-letter to the plaintext. Such an attack could be thwarted by using a *poly-alphabetic substitution cipher* where the key instead defines a mapping that is applied on *blocks* of plaintext characters. Here, for example, a key might map the 2-character block `ab` to `DZ` while mapping `ac` to `TY`; note that the plaintext character `a` does not get mapped to a fixed ciphertext character. Poly-alphabetic substitution ciphers "smooth out" the frequency distribution of characters in the ciphertext and make it harder to perform statistical analysis.

The *Vigenère cipher*, a poly-alphabetic shift cipher that is a special case of the above, can be viewed as applying different instances of the shift cipher to different parts of the plaintext. The key is now viewed as a *string* of letters; encryption is done by shifting each plaintext character by the amount indicated by the next character of the key, wrapping around in the key when necessary. (This degenerates to the shift cipher if the key has length 1.) For example, encryption of the message `tellhimaboutme` using the key `cafe` would work as follows:

Plaintext:	`tellhimaboutme`
Key (repeated):	`cafecafecafeca`
Ciphertext:	`VEQPJIREDOZXOE`

(The key need not be an English word.) This is exactly the same as encrypting the first, fifth, ninth, ... characters with the shift cipher and key `c`; the second, sixth, tenth, ... characters with key `a`; the third, seventh, ... characters with `f`; and the fourth, eighth, ... characters with `e`. Notice that in the above example `l` is mapped once to `Q` and once to `P`. Furthermore, the ciphertext character `E` is sometimes obtained from `e` and sometimes from `a`. Thus, the character frequencies of the ciphertext are "smoothed out," as desired.

If the key is sufficiently long, cracking this cipher appears daunting. Indeed, it had been considered by many to be "unbreakable," and although it was invented in the 16th century, a systematic attack on the scheme was only devised hundreds of years later.

Attacking the Vigenère cipher. A first observation in attacking the Vigenère cipher is that *if the length of the key is known* then attacking the cipher is relatively easy. Specifically, say the length of the key, also called the *period*, is t. Write the key k as $k = k_1 \cdots k_t$ where each k_i is a letter of the alphabet. An observed ciphertext $c = c_1 c_2 \cdots$ can be divided into t parts where each part can be viewed as having been encrypted using the shift

cipher. Specifically, for all $j \in \{1, \ldots, t\}$ the ciphertext characters

$$c_j, c_{j+t}, c_{j+2t}, \ldots$$

all resulted by shifting the corresponding characters of the plaintext by k_j positions. We refer to the above sequence of characters as the jth *stream*. All that remains is to determine, for each of the t streams, which of the 26 possible shifts was used. This is not as trivial as in the case of the shift cipher, because it is no longer possible to simply try different shifts in an attempt to determine when decryption of a stream "makes sense." (Recall that a stream does not correspond to consecutive letters in the plaintext.) Furthermore, trying to guess the entire key k at once would require a brute-force search through 26^t different possibilities, which is infeasible for large t. Nevertheless, we can still use letter-frequency analysis to analyze each stream independently. Namely, for each stream we tabulate the frequency of each ciphertext character and then check which of the 26 possible shifts yields the "right" probability distribution for that stream. Since this can be carried out independently for each stream (i.e., for each character of the key), this attack takes time $26 \cdot t$ rather than time 26^t.

A more principled, easier-to-automate approach is to apply the improved attack on the shift cipher (discussed earlier) to each stream. That attack did not rely on checking for a plaintext that "made sense," but only relied on the underlying frequency distribution of characters in the plaintext.

Either of the above approaches gives a successful attack when the key length is known. What if the key length is unknown?

Note first that as long as the maximum length T of the key is not too large, we can simply repeat the above attack T times (once for each possible value $t \in \{1, \ldots, T\}$). This leads to at most T different candidate plaintexts, among which the true plaintext will likely be easy to identify. So an unknown key length is not a serious obstacle.

There are also more efficient ways to determine the key length from an observed ciphertext. One is to use *Kasiski's method*, published in the mid-19th century. The first step here is to identify repeated patterns of length 2 or 3 in the ciphertext. These are likely the result of certain bigrams or trigrams that appear frequently in the plaintext. For example, consider the common word "`the`." This word will be mapped to different ciphertext characters, depending on its position in the plaintext. However, if it appears twice in the same relative position, then it will be mapped to the same ciphertext characters. For a sufficiently long plaintext, there is thus a good chance that "`the`" will be mapped repeatedly to the same ciphertext characters.

Consider the following concrete example with the key `beads` (spaces have been added for clarity):

Plaintext:	`the man and the woman retrieved the letter from the post office`
Key:	`bea dsb ead sbe adsbe adsbeadsb ead sbeads bead sbe adsb eadsbe`
Ciphertext:	`ULE PSO ENG LII WREBR RHLSMEYWE XHH DFXTHJ GVOP LII PRKU SFIADI`

The word `the` is mapped sometimes to `ULE`, sometimes to `LII`, and sometimes to `XHH`. However, it is mapped *twice* to `LII`, and in a long enough text it is likely that it would be mapped multiple times to each possibility. Kasiski's observation was that the distance between such repeated appearances (assuming they are not coincidental) is a multiple of the period. (In the above example, the period is 5 and the distance between the two appearances of `LII` is 30, which is 6 times the period.) Therefore, the greatest common divisor of the distances between repeated sequences (assuming they are not coincidental) will yield the key length t or a multiple thereof.

An alternative approach, called the *index of coincidence method*, is more methodical and hence easier to automate. Recall that if the key length is t, then the ciphertext characters

$$c_1, c_{1+t}, c_{1+2t}, \ldots$$

in the first stream all resulted from encryption using the same shift. This means that the frequencies of the characters in this sequence are expected to be identical to the character frequencies of standard English text *in some shifted order*. In more detail: let q_i denote the observed frequency of the ith letter in this stream; this is simply the number of occurrences of the ith letter of the alphabet divided by the total number of letters in the stream. If the shift used here is j (i.e., if the first character k_1 of the key is equal to j), then for all i we expect $q_{i+j} \approx p_i$, where p_i is the frequency of the ith letter of the alphabet in standard English text. (Once again, we use q_{i+j} in place of $q_{[i+j \bmod 26]}$.) But this means that the sequence $q_0, \ldots, q_{25}$ is just the sequence $p_0, \ldots, p_{25}$ shifted j places. As a consequence (cf. Equation (1.1)):

$$\sum_{i=0}^{25} q_i^2 \approx \sum_{i=0}^{25} p_i^2 \approx 0.065.$$

This leads to a nice way to determine the key length t. For $\tau = 1, 2, \ldots, T$, look at the sequence of ciphertext characters $c_1, c_{1+\tau}, c_{1+2\tau}, \ldots$ and tabulate $q_0, \ldots, q_{25}$ for this sequence. Then compute

$$S_\tau \stackrel{\text{def}}{=} \sum_{i=0}^{25} q_i^2. \tag{1.3}$$

When $\tau = t$ we expect $S_\tau \approx 0.065$, as discussed above. On the other hand, if τ is not a multiple of t we expect that all characters will occur with roughly equal probability in the sequence $c_1, c_{1+\tau}, c_{1+2\tau}, \ldots$, and so we expect $q_i \approx 1/26$ for all i. In this case we will obtain

$$S_\tau \approx \sum_{i=0}^{25} \left(\frac{1}{26}\right)^2 \approx 0.038.$$

The smallest value of τ for which $S_\tau \approx 0.065$ is thus likely the key length. One can further validate a guess τ by carrying out a similar calculation using the second stream $c_2, c_{2+\tau}, c_{2+2\tau}, \ldots$, etc.

Ciphertext length and cryptanalytic attacks. The above attacks on the Vigenère cipher require a longer ciphertext than the attacks on previous schemes. For example, the index of coincidence method requires c_1, c_{1+t}, c_{1+2t} (where t is the actual key length) to be sufficiently long in order to ensure that the observed frequencies are close to what is expected; the ciphertext itself must then be roughly t times larger. Similarly, the attack we showed on the mono-alphabetic substitution cipher requires a longer ciphertext than the attack on the shift cipher (which can work for encryptions of even a single word). This illustrates that a longer key can, in general, require the cryptanalyst to obtain more ciphertext in order to carry out an attack. (Indeed, the Vigenère cipher can be shown to be secure if the key is as long as what is being encrypted. We will see a related phenomenon in the next chapter.)

Conclusions. We have presented only a few historical ciphers. Beyond their historical interest, our aim in presenting them was to illustrate some important lessons. Perhaps the most important is that *designing secure ciphers is hard.* The Vigenère cipher remained unbroken for a long time. Far more complex schemes have also been used. But a complex scheme is not necessarily secure, and all historical schemes have been broken.

1.4 Principles of Modern Cryptography

As should be clear from the previous section, cryptography was historically more of an art than a science. Schemes were designed in an heuristic manner and evaluated based on their perceived complexity or cleverness. A scheme would be analyzed to see if any attacks could be found; if so, the scheme would be "patched" to thwart that attack, and the process repeated. Although there may have been agreement that some schemes were *not* secure (as evidenced by an especially damaging attack), there was no agreed-upon notion of what requirements a "secure" scheme should satisfy, and no way to give evidence that any specific scheme *was* secure.

Over the past several decades, cryptography has developed into more of a science. Schemes are now developed and analyzed in a more systematic manner, with the ultimate goal being to give a rigorous *proof* that a given construction is secure. In order to articulate such proofs, we first need *formal definitions* that pin down exactly what "secure" means; such definitions are useful and interesting in their own right. As it turns out, most cryptographic proofs rely on currently unproven *assumptions* about the algorithmic hardness of certain mathematical problems; any such assumptions must be made

explicit and be stated precisely. An emphasis on definitions, assumptions, and proofs distinguishes *modern* cryptography from classical cryptography; we now discuss these three principles in greater detail.

1.4.1 Principle 1 – Formal Definitions

One of the key contributions of modern cryptography has been the recognition that formal definitions of security are *essential* for the proper design, study, evaluation, and usage of cryptographic primitives. Put bluntly:

> *If you don't understand what you want to achieve, how can you possibly know when (or if) you have achieved it?*

Formal definitions provide such understanding by giving a clear description of what threats are in scope and what security guarantees are desired. As such, definitions can help guide the design of cryptographic schemes. Indeed, it is much better to formalize what is required *before* the design process begins, rather than to come up with a definition *post facto* once the design is complete. The latter approach risks having the design phase end when the designers' patience is exhausted (rather than when the goal has been met), or may result in a construction achieving *more* than is needed at the expense of efficiency.

Definitions also offer a way to evaluate and analyze constructions. With a definition in place, one can study a proposed scheme to see if it achieves the desired guarantees; in some cases, one can even *prove* a given construction secure (see Section 1.4.3) by showing that it meets the definition. On the flip side, definitions can be used to conclusively show that a given scheme is *not* secure, insofar as the scheme does not satisfy the definition. In particular, observe that the attacks in the previous section do not conclusively demonstrate that any of the schemes shown there is "insecure." For example, the attack on the Vigenère cipher assumed that sufficiently long English text was being encrypted, but perhaps the Vigenère cipher is "secure" if short English text, or compressed text (which will have roughly uniform letter frequencies), is encrypted? It is hard to say without a formal definition in place.

Definitions enable a meaningful comparison of schemes. As we will see, there can be *multiple* (valid) ways to define security; the "right" one depends on the context in which a scheme is used. A scheme satisfying a weaker definition may be more efficient than another scheme satisfying a stronger definition; with precise definitions we can properly evaluate the trade-offs between the two schemes. Along the same lines, definitions enable secure usage of schemes. Consider deciding which encryption scheme to use for some larger application. A sound way to approach the problem is to first understand what notion of security is required for that application, and then find an encryption scheme satisfying that notion. A side benefit of this approach is *modularity*: a designer can "swap out" one encryption scheme and replace it with another (that also satisfies the necessary definition of security) without having to worry about affecting security of the overall application.

Writing a formal definition forces one to think about what is essential to the problem at hand and what properties are extraneous. Going through the process often reveals subtleties of the problem that were not obvious at first glance. We illustrate this next for the case of encryption.

An example: secure encryption. A common mistake is to think that formal definitions are not needed, or are trivial to come up with, because "everyone has an intuitive idea of what security means." This is not the case. As an example, we consider the case of encryption. (The reader may want to pause here to think about how they would formally define what it means for an encryption scheme to be secure.) Although we postpone a formal definition of secure encryption to subsequent chapters, we describe here informally what such a definition should capture.

In general, a security definition has two components: a security guarantee (or, from the attacker's point of view, what constitutes a successful attack) and a threat model. The security guarantee defines what the scheme is intended to prevent the attacker from doing, while the threat model describes the power of the adversary, i.e., what actions the attacker is assumed able to carry out.

Let's start with the first of these. What should a secure encryption scheme guarantee? Here are some thoughts:

- *It should be impossible for an attacker to recover the key.* We have previously observed that if an attacker can determine the key shared by two parties using some scheme, then that scheme cannot be secure. However, it is easy to come up with schemes for which key recovery is impossible, yet the scheme is blatantly insecure. Consider, e.g., the scheme where $\mathsf{Enc}_k(m) = m$. The ciphertext leaks no information about the key (and so the key cannot be recovered if it is long enough) yet the message is sent in the clear! We thus see that inability to recover the key is necessary but not sufficient for security. This makes sense: the aim of encryption is to protect the *message*; secrecy of the key is a means for achieving this goal, but is not itself the objective.

- *It should be impossible for an attacker to recover the plaintext from the ciphertext.* This definition is better, but is still far from satisfactory. In particular, this definition would consider an encryption scheme secure if its ciphertexts revealed 90% of the plaintext, as long as 10% of the plaintext remained hard to figure out. This is clearly unacceptable in most common applications of encryption; for example, when encrypting a salary database, we would be justifiably upset if 90% of employees' salaries were revealed!

- *It should be impossible for an attacker to recover any character of the plaintext from the ciphertext.* This looks like a good definition, yet is still not sufficient. Going back to the example of encrypting a salary database, we would not consider an encryption scheme secure if it reveals whether an employee's salary is more than or less than \$100,000,

even if it does not reveal any particular digit of that employee's salary. Similarly, we would not want an encryption scheme to reveal whether one particular employee makes more than another.

Another issue is how to formalize what it means for an adversary to "recover a character of the plaintext." What if an attacker correctly guesses, through sheer luck or external information, that the least significant digit of someone's salary is 0? Clearly that should not render an encryption scheme insecure, and so any viable definition must somehow rule out such behavior from qualifying as a successful attack.

- *The "right" answer: regardless of any information an attacker already has, a ciphertext should leak no* additional *information about the underlying plaintext.* This informal definition captures all the concerns outlined above. Note in particular that it does not try to define what information about the plaintext is "meaningful"; it simply requires that *no* information be leaked. This is important, as it means that a secure encryption scheme is suitable for all potential applications in which secrecy is required.

  What is missing here is a precise, mathematical formulation of the definition. How should we capture an attacker's prior knowledge about the plaintext? And what does it mean to (not) leak information? We will return to these questions in the next two chapters; see especially Definitions 2.3 and 3.12.

Now that we have fixed a security *goal*, it remains to specify a *threat model*. This specifies what "power" the attacker is assumed to have, but does not place any restrictions on the adversary's *strategy*. This is an important distinction: we specify what we assume about the adversary's abilities, but we do *not* assume anything about *how it uses* those abilities. It is impossible to foresee what strategies might be used in an attack, and history has proven that attempts to do so are doomed to failure.

There are several plausible options for the threat model in the context of encryption; standard ones, in order of increasing power of the attacker, are:

- **Ciphertext-only attack:** This is the most basic attack, where the adversary just observes a ciphertext (or multiple ciphertexts) and attempts to determine information about the underlying plaintext (or plaintexts). This is the threat model we have been implicitly assuming when discussing classical encryption schemes in the previous section.

- **Known-plaintext attack:** Here, the adversary is able to learn one or more plaintext/ciphertext pairs generated using some key. The aim of the adversary is then to deduce information about the underlying plaintext of some *other* ciphertext produced using the same key.

  All the classical encryption schemes we have seen are trivial to break using a known-plaintext attack; we leave a demonstration as an exercise.

- **Chosen-plaintext attack:** In this attack, the adversary can obtain plaintext/ciphertext pairs, as above, for plaintexts *of its choice*.

- **Chosen-ciphertext attack:** The final type of attack is one where the adversary is additionally able to obtain (some information about) the *decryption* of ciphertexts of its choice, e.g., whether the decryption of some ciphertext chosen by the attacker yields a valid English message. The adversary's aim, once again, is to learn information about the underlying plaintext of some *other* ciphertext (whose decryption the adversary is unable to obtain directly) generated using the same key.

Although the threat models are listed in order of increasing strength, none of them is inherently better than any other; the right one to use depends on the environment in which an encryption scheme is deployed.

The first two types of attack are the easiest to carry out. In a ciphertext-only attack, the only thing the adversary needs to do is eavesdrop on the communication channel over which encrypted messages are sent. In a known-plaintext attack it is assumed the adversary also obtains ciphertexts corresponding to known plaintexts. This is often easy to accomplish because not all encrypted messages are secret, at least not indefinitely. As a trivial example, two parties may always encrypt a "hello" message whenever they begin communicating. As a more complex example, encryption may be used to keep quarterly-earnings reports secret until their release date; in this case, anyone eavesdropping on the ciphertext will later obtain the corresponding plaintext.

In the latter two attacks the adversary is assumed to be able to obtain encryptions and/or decryptions of plaintexts/ciphertexts of its choice. This may at first seem strange, and we defer a more detailed discussion of these attacks, and their practicality, to Section 3.4.2 (for chosen-plaintext attacks) and Section 5.1 (for chosen-ciphertext attacks).

1.4.2 Principle 2 – Precise Assumptions

Most modern cryptographic constructions cannot be proven secure unconditionally; such proofs would require resolving questions in the theory of computational complexity that seem far from being answered today.[4] The result of this unfortunate state of affairs is that proofs of security typically rely on *assumptions*. Modern cryptography requires any such assumptions to be made explicit and mathematically precise. At the most basic level, this is because proofs of security require this. But there are other reasons as well:

1. *Validation of assumptions:* By their very nature, assumptions are statements that are not proven but are instead conjectured to be true. In order to strengthen our belief in some assumption, it is necessary to

[4] In particular, most of cryptography requires the unproven assumption that $\mathcal{P} \neq \mathcal{NP}$.

study it: The more the assumption is examined and tested without being refuted, the more confident we are that the assumption is true. Furthermore, study of an assumption can provide evidence of its validity by showing that it is implied by some other assumption that is also widely believed.

If the assumption being relied upon is not precisely stated, it cannot be effectively studied and (potentially) refuted. Thus, a precondition to increasing our confidence in an assumption is having a precise statement of what exactly is being assumed.

2. *Comparison of assumptions:* Often in cryptography we are presented with two schemes that can both be proven to satisfy some definition, each based on a different assumption. Assuming all else is equal, which scheme should be preferred? If the assumption on which the first scheme is based is *weaker* than the assumption on which the second scheme is based (i.e., if the second assumption implies the first), then the first scheme is preferable since it may turn out that the second assumption is false while the first assumption is true. If the assumptions used by the two schemes are not comparable, then the general rule is to prefer the scheme that is based on the better-studied assumption in which there is presumably greater confidence.

3. *Understanding the necessary assumptions:* An encryption scheme may be based on some underlying building block. If some weaknesses are later found in the building block, how can we tell whether the encryption scheme is still secure? If the underlying assumptions regarding the building block are made clear as part of proving security of the scheme, then we need only check whether the required assumptions are affected by the new weaknesses that were found.

A question that sometimes arises is: rather than prove a scheme secure based on some other assumption, why not simply assume that the scheme *itself* is secure? In some cases—e.g., when the definition is simple and a scheme has successfully resisted attack for many years—this may be an acceptable approach. But this approach is not preferred, and is downright dangerous when a new construction is being introduced. The reasons above help explain why. First, an assumption that has been studied for several years is preferable to a new, arbitrary assumption that is introduced along with a new construction. Second, there is a general preference for "simpler" assumptions—i.e., an assumption about the hardness of a clean mathematical problem vs. an assumption that a complex scheme satisfies an elaborate security definition—since simpler assumptions are in general easier to understand and study. Another advantage of relying on "lower-level" assumptions (rather than just assuming a scheme is secure) is that these low-level assumptions can typically be used in other constructions. Finally, low-level assumptions enable *modularity*. Consider an encryption scheme whose security relies on some assumed property

of one of its building blocks. If the underlying building block turns out *not* to satisfy the stated assumption, the encryption scheme can be instantiated using a different component that satisfies the necessary requirements.

1.4.3 Principle 3 – Proofs of Security

The two principles just described allow us to achieve our goal of providing rigorous *proof* that a construction satisfies a given definition under certain assumptions. Such proofs are especially important in the context of cryptography where there is an attacker who is actively trying to "break" some scheme. Proofs of security give an iron-clad guarantee—relative to the definition and assumptions—that no attacker will succeed; this is much better than taking an unprincipled or heuristic approach to the problem. Without a proof that no adversary with the specified resources can break some scheme, we are left only with our intuition that this is the case. Experience has shown that intuition in cryptography and computer security is disastrous. There are countless examples of unproven schemes that were broken, sometimes immediately and sometimes years after being developed.

Summary: Rigorous vs. Heuristic Approaches to Security

Reliance on definitions, assumptions, and proofs constitutes a rigorous approach to cryptography that is distinct from the informal approach of classical cryptography. Unfortunately, unprincipled, "off-the-cuff" solutions are still designed and deployed by those wishing to obtain a quick solution to a problem, or by those who are simply unknowledgable. We hope this book will contribute to an awareness of the rigorous approach and its importance in developing provably secure schemes.

1.4.4 Provable Security and Real-World Security

Much of modern cryptography now rests on sound mathematical foundations. But this does not mean that the field is no longer partly an *art* as well. The rigorous approach leaves room for creativity in developing definitions suited to contemporary applications and environments, in proposing new mathematical assumptions and designing new primitives, and in constructing novel schemes and proving them secure. There will also always be the art of *attacking* deployed cryptosystems, even when they are proven secure. We expand on this point next.

The approach taken by modern cryptography has revolutionized the field, and helps provide confidence in the security of cryptographic schemes deployed in the real world. But it is important not to overstate what a proof of security implies. A proof of security is always relative to the *definition* being considered and the *assumption(s)* being used. If the security guarantee does not match what is needed, or the threat model does not capture the adversary's true

abilities, then the proof may be irrelevant. Similarly, if the assumption that is relied upon turns out to be false, then the proof of security is meaningless.

The take-away point is that provable security of a scheme does not necessarily imply security of that scheme in the real world.[5] While some have viewed this as a drawback of provable security, we view this optimistically as illustrating the *strength* of the approach. To attack a provably secure scheme in the real world, the attacker is forced to focus attention on the definition (i.e., to explore how the idealized definition differs from the real-world requirements) or the underlying assumptions (i.e., to see whether they hold). In turn, it is the job of cryptographers to continually refine their definitions to more closely match the real world, and to investigate their assumptions to test their validity. Provable security does not end the age-old battle between attacker and defender, but it does provide a framework that helps shift the odds in the defender's favor.

References and Additional Reading

In this chapter, we have studied just a few of the known historical ciphers. There are many others of both historical and mathematical interest, and we refer the reader to textbooks by Stinson [195] or Trappe and Washington [196] for further details. The important role cryptography has played throughout history is a fascinating subject covered in books by Kahn [106] and Singh [188].

Shannon [177] was the first to pursue a rigorous approach to cryptography based on precise definitions and mathematical proofs; we explore his work in the next chapter.

Exercises

1.1 Decrypt the ciphertext provided at the end of the section on mono-alphabetic substitution ciphers.

1.2 Provide a formal definition of the Gen, Enc, and Dec algorithms for the mono-alphabetic substitution cipher.

1.3 Provide a formal definition of the Gen, Enc, and Dec algorithms for the Vigenère cipher. (Note: there are several plausible choices for Gen; choose one.)

[5] Here we are not even considering the possibility of an incorrect *implementation* of the scheme. Poorly implemented cryptography is a serious problem in the real world, but this problem is largely outside the scope of this book.

1.4 Say you are given a ciphertext that corresponds to English-language text that was encrypted using either the shift cipher or the Vigenère cipher with period greater than 1. How could you tell which was the case?

1.5 Implement the attacks described in this chapter for the shift cipher and the Vigenère cipher.

1.6 The shift and Vigenère ciphers can also be defined on the 256-character alphabet consisting of all possible bytes (8-bit strings), and using XOR instead of modular addition.

(a) Provide a formal definition of both schemes in this case.

(b) Discuss how the attacks we have shown in this chapter can be modified to break these schemes.

1.7 The index of coincidence method relies on a known value for the sum of the *squares* of plaintext-letter frequencies (cf. Equation (1.1)). Why would it not work using the sum $\sum_i p_i$ itself?

1.8 Show that the shift, substitution, and Vigenère ciphers are all trivial to break using a chosen-plaintext attack. How much chosen plaintext is needed to recover the key for each of the ciphers?

1.9 Assume an attacker knows that a user's password is either `abcd` or `bedg`. Say the user encrypts his password using the shift cipher, and the attacker sees the resulting ciphertext. Show how the attacker can determine the user's password, or explain why this is not possible.

1.10 Repeat the previous exercise for the Vigenère cipher using period 2, using period 3, and using period 4.

1.11 The attack on the Vigenère cipher has two steps: (a) find the key length by identifying τ with $S_\tau \approx 0.065$ (cf. Equation (1.3)) and (b) for each character of the key, find j maximizing I_j (cf. Equation (1.2)), using $\{p_i\}$ corresponding to English text. What happens in each case if the underlying plaintext is in a language other than English?

Chapter 2

Perfectly Secret Encryption

In the previous chapter we presented some historical encryption schemes and showed that they can be broken easily. In this chapter, we look at the other extreme and study encryption schemes that are *provably* secure even against an adversary with unbounded computational power. Such schemes are called *perfectly secret*. We rigorously define this notion, and explore conditions under which perfect secrecy can be achieved.

The material in this chapter belongs, in some sense, more to the world of "classical" cryptography than to the world of "modern" cryptography. Besides the fact that all the material introduced here was developed before the revolution in cryptography that took place in the mid-1970s and 1980s, the constructions we study in this chapter rely only on the first and third principles outlined in Section 1.4. That is, precise mathematical definitions are used and rigorous proofs are given, but it will not be necessary to rely on any unproven computational assumptions. It is clearly advantageous to avoid such assumptions; we will see, however, that doing so has inherent limitations. Thus, in addition to serving as a good basis for understanding the principles underlying modern cryptography, the results of this chapter also justify our later adoption of all three of the aforementioned principles.

Beginning with this chapter, we will define security and analyze schemes using probabilistic experiments involving randomized algorithms. (We assume familiarity with basic probability theory. The relevant notions are reviewed in Appendix A.3.) A simple example is given by the "experiment" in which the parties who wish to communicate using a private-key encryption scheme generate a random key. Since randomness is so essential, we briefly discuss the issue of generating randomness suitable for cryptographic applications before returning to a discussion of cryptography *per se*.

Generating randomness. Throughout the book, we will assume for simplicity that parties have access to an unlimited supply of independent, unbiased (i.e., uniform) bits. Where do these random bits come from? Since classical computation is deterministic, it is not at all clear how computers can be used to generate random bits. In principle, one could generate a small number of uniform bits by hand, e.g., by flipping a fair coin. But that approach is not very convenient, nor does it scale.

Modern *random-number generation* proceeds in two steps. First, a "pool" of high-entropy data is collected. (For our purposes a formal definition of

entropy is not needed, and it suffices to think of entropy as a measure of unpredictability.) Next, this high-entropy data is processed to yield a sequence of nearly independent and unbiased bits. This second step is necessary since high-entropy data is not necessarily uniform.

For the first step, some source of unpredictable data is needed. This can come from external inputs, for example, delays between network events, hard-disk access times, keystrokes or mouse movements made by the user, and so on. More sophisticated approaches—which, by design, incorporate random-number generation more tightly into the system at the hardware level—can also be used. These rely on physical phenomena such as thermal/shot noise or radioactive decay; for example, certain Intel processors use thermal noise to generate high-entropy data on-chip. Hardware random-number generators of this sort generally produce high-entropy data at a faster rate than techniques relying on external sources.

The processing needed to "smooth" the high-entropy data to obtain (nearly) independent and uniform bits is non-trivial, and is discussed briefly in Section 6.6.4. Here, we consider a simple example to give an idea of what can be done. Imagine that our high-entropy pool contains a sequence of *biased* bits, where 1 occurs with probability p and 0 occurs with probability $1-p$. (We do assume, however, that the bits are all *independent*. In practice this assumption is typically not valid and so more-complex processing must be done.) Thousands of such bits have lots of entropy, but are not close to uniform. We can obtain a uniform sequence of bits by taking the original bits in pairs: if we see a 1 followed by a 0 then we output 0, and if we see a 0 followed by a 1 then we output 1. (If we see two 0s or two 1s in a row we output nothing, and simply move on to the next pair.) The probability that any pair results in a 0 is $p \cdot (1-p)$, which is exactly equal to the probability that any pair results in a 1. (Note that we do not even need to know the value of p!) We thus obtain a uniformly distributed output from our initial high-entropy pool.

Care must be taken in how random bits are produced, and using poor random-number generators can often leave a good cryptosystem vulnerable to attack. One should use a random-number generator that is *designed for cryptographic use*, rather than a "general-purpose" random-number generator that is generally not suitable for cryptographic applications. In particular, the `rand()` function in the C stdlib.h library is *not* cryptographically secure, and using it in cryptographic settings can have disastrous consequences.

2.1 Definitions

We begin by recalling and expanding upon the syntax of encryption, as introduced in the previous chapter. An encryption scheme is defined by three

algorithms Gen, Enc, and Dec, as well as a specification of a *message space* $\mathcal{M}$ with $|\mathcal{M}| > 1$.[1] The key-generation algorithm Gen is a probabilistic algorithm that outputs a key k chosen according to some distribution. We denote by $\mathcal{K}$ the (finite) *key space*, i.e., the set of all possible keys that can be output by Gen. The encryption algorithm Enc takes as input a key $k \in \mathcal{K}$ and a message $m \in \mathcal{M}$, and outputs a ciphertext c. We now explicitly allow the encryption algorithm to be probabilistic (so $\mathsf{Enc}_k(m)$ might output a different ciphertext when run multiple times), and we write $c \leftarrow \mathsf{Enc}_k(m)$ to denote the possibly probabilistic process by which message m is encrypted using key k to give ciphertext c. (Looking ahead, we also sometimes use the notation $x \leftarrow S$ to denote uniform selection of x from a set S. In case Enc is deterministic, we may emphasize this by writing $c := \mathsf{Enc}_k(m)$.) We let $\mathcal{C}$ denote the set of all possible ciphertexts that can be output by $\mathsf{Enc}_k(m)$, for all possible choices of $k \in \mathcal{K}$ and $m \in \mathcal{M}$ (and for all random choices of Enc in case it is randomized). The decryption algorithm Dec takes as input a key $k \in \mathcal{K}$ and a ciphertext $c \in \mathcal{C}$ and outputs a message $m \in \mathcal{M}$. We assume *perfect correctness*, meaning that for all $k \in \mathcal{K}$, $m \in \mathcal{M}$, and any ciphertext c output by $\mathsf{Enc}_k(m)$, it holds that $\mathsf{Dec}_k(c) = m$ with probability 1. Perfect correctness implies that we may assume Dec is deterministic without loss of generality, since $\mathsf{Dec}_k(c)$ must give the same output every time it is run. We will thus write $m := \mathsf{Dec}_k(c)$ to denote the (deterministic) process of decrypting ciphertext c using key k to yield the message m.

In the definitions and theorems below, we refer to probability distributions over $\mathcal{K}$, $\mathcal{M}$, and $\mathcal{C}$. The distribution over $\mathcal{K}$ is the one defined by running Gen and taking the output. (It is almost always the case that Gen chooses a key uniformly from $\mathcal{K}$ and, in fact, we may assume this without loss of generality; see Exercise 2.1.) We let K be the random variable denoting the value of the key output by Gen; thus, for any $k \in \mathcal{K}$, $\Pr[K = k]$ denotes the probability that the key output by Gen is equal to k. Similarly, we let M be the random variable denoting the message being encrypted, so $\Pr[M = m]$ denotes the probability that the message takes on the value $m \in \mathcal{M}$. The probability distribution of the message is not determined by the encryption scheme itself, but instead reflects the likelihood of different messages being sent by the parties using the scheme, as well as an adversary's uncertainty about what will be sent. As an example, an adversary may know that the message will either be `attack today` or `don't attack`. The adversary may even know (by other means) that with probability 0.7 the message will be a command to attack and with probability 0.3 the message will be a command not to attack. In this case, we have $\Pr[M = \texttt{attack today}] = 0.7$ and $\Pr[M = \texttt{don't attack}] = 0.3$.

K and M are required to be independent, i.e., what is being communicated by the parties must be independent of the key they share. This makes sense,

[1] If $|\mathcal{M}| = 1$ there is only one message and no point in communicating, let alone encrypting.

among other reasons, because the distribution over $\mathcal{K}$ is determined by the encryption scheme itself (since it is defined by Gen), while the distribution over $\mathcal{M}$ depends on the context in which the encryption scheme is being used.

Fixing an encryption scheme and a distribution over $\mathcal{M}$ determines a distribution over the space of ciphertexts $\mathcal{C}$ given by choosing a key $k \in \mathcal{K}$ (according to Gen) and a message $m \in \mathcal{M}$ (according to the given distribution), and then computing the ciphertext $c \leftarrow \mathsf{Enc}_k(m)$. We let C be the random variable denoting the resulting ciphertext and so, for $c \in \mathcal{C}$, write $\Pr[C = c]$ to denote the probability that the ciphertext is equal to the fixed value c.

Example 2.1

We work through a simple example for the shift cipher (cf. Section 1.3). Here, by definition, we have $\mathcal{K} = \{0, \ldots, 25\}$ with $\Pr[K = k] = 1/26$ for each $k \in \mathcal{K}$.

Say we are given the following distribution over $\mathcal{M}$:

$$\Pr[M = \mathtt{a}] = 0.7 \quad \text{and} \quad \Pr[M = \mathtt{z}] = 0.3.$$

What is the probability that the ciphertext is $\mathtt{B}$? There are only two ways this can occur: either $M = \mathtt{a}$ and $K = 1$, or $M = \mathtt{z}$ and $K = 2$. By independence of M and K, we have

$$\begin{aligned}\Pr[M = \mathtt{a} \wedge K = 1] &= \Pr[M = \mathtt{a}] \cdot \Pr[K = 1] \\ &= 0.7 \cdot \left(\frac{1}{26}\right).\end{aligned}$$

Similarly, $\Pr[M = \mathtt{z} \wedge K = 2] = 0.3 \cdot \left(\frac{1}{26}\right)$. Therefore,

$$\begin{aligned}\Pr[C = \mathtt{B}] &= \Pr[M = \mathtt{a} \wedge K = 1] + \Pr[M = \mathtt{z} \wedge K = 2] \\ &= 0.7 \cdot \left(\frac{1}{26}\right) + 0.3 \cdot \left(\frac{1}{26}\right) \;=\; 1/26.\end{aligned}$$

We can calculate conditional probabilities as well. For example, what is the probability that the message $\mathtt{a}$ was encrypted, given that we observe ciphertext $\mathtt{B}$? Using Bayes' Theorem (Theorem A.8) we have

$$\begin{aligned}\Pr[M = \mathtt{a} \mid C = \mathtt{B}] &= \frac{\Pr[C = \mathtt{B} \mid M = \mathtt{a}] \cdot \Pr[M = \mathtt{a}]}{\Pr[C = \mathtt{B}]} \\ &= \frac{\Pr[C = \mathtt{B} \mid M = \mathtt{a}] \cdot 0.7}{1/26}.\end{aligned}$$

Note that $\Pr[C = \mathtt{B} \mid M = \mathtt{a}] = 1/26$, since if $M = \mathtt{a}$ then the only way $C = \mathtt{B}$ can occur is if $K = 1$ (which occurs with probability $1/26$). We conclude that $\Pr[M = \mathtt{a} \mid C = \mathtt{B}] = 0.7$. ♢

Example 2.2

Consider the shift cipher again, but with the following distribution over $\mathcal{M}$:

$$\Pr[M = \mathtt{kim}] = 0.5, \; \Pr[M = \mathtt{ann}] = 0.2, \; \Pr[M = \mathtt{boo}] = 0.3.$$

What is the probability that $C = \texttt{DQQ}$? The only way this ciphertext can occur is if $M = \texttt{ann}$ and $K = 3$, or $M = \texttt{boo}$ and $K = 2$, which happens with probability $0.2 \cdot 1/26 + 0.3 \cdot 1/26 = 1/52$.

We can also compute the probability that `ann` was encrypted, conditioned on observing the ciphertext `DQQ`? A calculation as above using Bayes' Theorem gives $\Pr[M = \texttt{ann} \mid C = \texttt{DQQ}] = 0.4$. ♢

Perfect secrecy. We are now ready to define the notion of *perfect secrecy*. We imagine an adversary who knows the probability distribution of M; that is, the adversary knows the likelihood that different messages will be sent. The adversary also knows the encryption scheme being used. The only thing unknown to the adversary is the key shared by the parties. A message is chosen by one of the honest parties and encrypted, and the resulting ciphertext is transmitted to the other party. The adversary can *eavesdrop* on the parties' communication, and thus observe this ciphertext. (That is, this is a ciphertext-only attack, where the attacker sees only a single ciphertext.) For a scheme to be perfectly secret, observing this ciphertext should have *no effect* on the adversary's knowledge regarding the actual message that was sent; in other words, the *a posteriori* probability that some message $m \in \mathcal{M}$ was sent, conditioned on the ciphertext that was observed, should be no different from the *a priori* probability that m would be sent. This means that the ciphertext reveals nothing about the underlying plaintext, and the adversary learns absolutely nothing about the plaintext that was encrypted. Formally:

DEFINITION 2.3 *An encryption scheme* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *with message space* $\mathcal{M}$ *is* perfectly secret *if for every probability distribution for* M*, every message* $m \in \mathcal{M}$*, and every ciphertext* $c \in \mathcal{C}$ *for which* $\Pr[C = c] > 0$*:*

$$\Pr[M = m \mid C = c] = \Pr[M = m].$$

(The requirement that $\Pr[C = c] > 0$ is a technical one needed to prevent conditioning on a zero-probability event.)

Example 2.4

We show that the shift cipher is *not* perfectly secret when used with the message space $\mathcal{M}$ consisting of all two-character plaintexts. To do so, we work with Definition 2.3, and show a probability distribution over $\mathcal{M}$ for which, for some message m and ciphertext c,

$$\Pr[M = m \mid C = c] \neq \Pr[M = m].$$

Many such distributions are possible, but we pick a simple one: say the message is either `aa` or `ab`, each with half probability. Set $m = \texttt{ab}$ and $c = \texttt{XX}$. Then clearly $\Pr[M = \texttt{ab} \mid C = \texttt{XX}] = 0$, as there is no way that `XX` can ever result from the encryption of `ab`. But $\Pr[M = \texttt{ab}] = 1/2$. ♢

We now give an equivalent formulation of perfect secrecy. This formulation defines perfect secrecy by requiring that the distribution of the ciphertext does not depend on the plaintext, i.e., for any two messages $m, m' \in \mathcal{M}$ the distribution of the ciphertext when m is encrypted should be identical to the distribution of the ciphertext when m' is encrypted. That is, for every $m, m' \in \mathcal{M}$, and every $c \in \mathcal{C}$, we have

$$\Pr[\mathsf{Enc}_K(m) = c] = \Pr[\mathsf{Enc}_K(m') = c] \tag{2.1}$$

(where the probabilities are over choice of K and any randomness of Enc). Note that the above probabilities depend *only* on the encryption scheme, and make no reference to any underlying distribution on $\mathcal{M}$. The above condition implies that a ciphertext contains no information about the plaintext, and that it is impossible to distinguish an encryption of m from an encryption of m', since the distributions of the ciphertext are the same in each case.

LEMMA 2.5 *An encryption scheme* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *with message space* $\mathcal{M}$ *is perfectly secret if and only if Equation (2.1) holds for every* $m, m' \in \mathcal{M}$ *and every* $c \in \mathcal{C}$.

PROOF The proof is straightforward, but we go through it in detail. The key observation is that for any scheme, any distribution on $\mathcal{M}$, any $m \in \mathcal{M}$ for which $\Pr[M = m] > 0$, and any $c \in \mathcal{C}$, we have

$$\begin{aligned} \Pr[C = c \mid M = m] &= \Pr[\mathsf{Enc}_K(M) = c \mid M = m] \\ &= \Pr[\mathsf{Enc}_K(m) = c \mid M = m] \\ &= \Pr[\mathsf{Enc}_K(m) = c], \end{aligned} \tag{2.2}$$

where the first equality is by definition of the random variable C, the second is because we are conditioning on the event that $M = m$, and the third is because K is independent of M. We also use the fact that for any $c \in \mathcal{C}$ with $\Pr[C = c] > 0$, we have

$$\Pr[M = m \mid C = c] \cdot \Pr[C = c] = \Pr[C = c \mid M = m] \cdot \Pr[M = m]. \tag{2.3}$$

Take the uniform distribution over $\mathcal{M}$. If the scheme is perfectly secret then $\Pr[M = m \mid C = c] = \Pr[M = m]$, and so Equation (2.3) implies that $\Pr[C = c \mid M = m] = \Pr[C = c]$. Since m and c were arbitrary, this shows that for every $m, m' \in \mathcal{M}$ and every $c \in \mathcal{C}$,

$$\begin{aligned} \Pr[\mathsf{Enc}_K(m) = c] &= \Pr[C = c \mid M = m] \\ &= \Pr[C = c] \\ &= \Pr[C = c \mid M = m'] = \Pr[\mathsf{Enc}_K(m') = c] \end{aligned}$$

(using Equation (2.2)), proving one direction of the lemma.

Conversely, say Equation (2.1) holds for every $m, m' \in \mathcal{M}$ and every $c \in \mathcal{C}$. Fix some distribution over $\mathcal{M}$, a message $m \in \mathcal{M}$, and a ciphertext $c \in \mathcal{C}$ with $\Pr[C = c] > 0$. If $\Pr[M = m] = 0$ then we trivially have

$$\Pr[M = m \mid C = c] = \Pr[M = m] = 0.$$

So, assume $\Pr[M = m] > 0$. For $c \in \mathcal{C}$, define $p_c \stackrel{\text{def}}{=} \Pr[\mathsf{Enc}_K(m) = c]$. Equations (2.1) and (2.2) imply that $\Pr[C = c \mid M = m'] = p_c$ for every $m' \in \mathcal{M}$. So,

$$\begin{aligned}\Pr[C = c] &= \sum_{m' \in \mathcal{M}} \Pr[C = c \mid M = m'] \cdot \Pr[M = m'] \\ &= \sum_{m' \in \mathcal{M}} p_c \cdot \Pr[M = m'] = p_c = \Pr[C = c \mid M = m],\end{aligned}$$

where the sum is over m' with $\Pr[M = m'] > 0$. Equation (2.3) implies that $\Pr[M = m \mid C = c] = \Pr[M = m]$, so the scheme is perfectly secret. ■

Perfect (adversarial) indistinguishability. We conclude this section by presenting another equivalent definition of perfect secrecy. This definition is based on an *experiment* involving an adversary passively observing a ciphertext and then trying to guess which of two possible messages was encrypted. We introduce this notion since it will serve as our starting point for defining computational security in the next chapter; throughout the rest of the book we will often use experiments like this one to define security.

In the present context, we consider the following experiment: An adversary $\mathcal{A}$ first specifies two arbitrary messages $m_0, m_1 \in \mathcal{M}$. Next, a key k is generated using Gen. Then, one of the two messages specified by $\mathcal{A}$ is chosen (each with probability 1/2) and encrypted using k; the resulting ciphertext is given to $\mathcal{A}$. Finally, $\mathcal{A}$ outputs a "guess" as to which of the two messages was encrypted; $\mathcal{A}$ *succeeds* if it guesses correctly. An encryption scheme is *perfectly indistinguishable* if *no* adversary $\mathcal{A}$ can succeed with probability better than 1/2. (Note that, for any encryption scheme, $\mathcal{A}$ can succeed with probability 1/2 by outputting a uniform guess; the requirement is simply that no attacker can do any better than this.) We stress that no limitations are placed on the computational power of $\mathcal{A}$.

Formally, let $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be an encryption scheme with message space $\mathcal{M}$. Let $\mathcal{A}$ be an adversary, which is formally just a (stateful) algorithm that we may assume is deterministic without loss of generality. We define an experiment $\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi}$, based, on $\mathcal{A}$ and Π, as follows:

The adversarial indistinguishability experiment $\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi}$:

1. *The adversary $\mathcal{A}$ outputs a pair of messages $m_0, m_1 \in \mathcal{M}$.*
2. *A key k is generated using Gen, and a uniform bit $b \in \{0,1\}$ is chosen. Ciphertext $c \leftarrow \mathsf{Enc}_k(m_b)$ is computed and given to $\mathcal{A}$. We refer to c as the* challenge ciphertext.

3. *$\mathcal{A}$ outputs a bit b'.*
4. *The output of the experiment is defined to be 1 if $b' = b$, and 0 otherwise. We write $\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi} = 1$ if the output of the experiment is 1 and in this case we say that $\mathcal{A}$ succeeds.*

As noted earlier, it is trivial for $\mathcal{A}$ to succeed with probability 1/2 by outputting a random guess. Perfect indistinguishability requires that it is impossible for any $\mathcal{A}$ to do better.

DEFINITION 2.6 *Encryption scheme $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ with message space $\mathcal{M}$ is* perfectly indistinguishable *if for every $\mathcal{A}$ it holds that*

$$\Pr\left[\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi} = 1\right] = \frac{1}{2}.$$

The following lemma states that Definition 2.6 is equivalent to Definition 2.3. We leave the proof of the lemma as Exercise 2.6.

LEMMA 2.7 *Encryption scheme Π is perfectly secret if and only if it is perfectly indistinguishable.*

Example 2.8
We show that the Vigenère cipher is *not* perfectly indistinguishable, at least for certain parameters. Concretely, let Π denote the Vigenère cipher for the message space of two-character strings, and where the period is chosen uniformly in $\{1, 2\}$. To show that Π is not perfectly indistinguishable, we exhibit an adversary $\mathcal{A}$ for which $\Pr\left[\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi} = 1\right] > \frac{1}{2}$.

Adversary $\mathcal{A}$ does:

1. Output $m_0 = \mathtt{aa}$ and $m_1 = \mathtt{ab}$.
2. Upon receiving the challenge ciphertext $c = c_1c_2$, do the following: if $c_1 = c_2$ output 0; else output 1.

Computation of $\Pr\left[\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi} = 1\right]$ is tedious but straightforward.

$$\begin{aligned}
&\Pr\left[\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi} = 1\right] \\
&= \frac{1}{2} \cdot \Pr\left[\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi} = 1 \mid b = 0\right] + \frac{1}{2} \cdot \Pr\left[\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi} = 1 \mid b = 1\right] \\
&= \frac{1}{2} \cdot \Pr[\mathcal{A} \text{ outputs } 0 \mid b = 0] + \frac{1}{2} \cdot \Pr[\mathcal{A} \text{ outputs } 1 \mid b = 1], \qquad (2.4)
\end{aligned}$$

where b is the uniform bit determining which message gets encrypted. As defined, $\mathcal{A}$ outputs 0 if and only if the two characters of the ciphertext $c = c_1c_2$ are equal. When $b = 0$ (so $m_0 = \mathtt{aa}$ is encrypted) then $c_1 = c_2$ if either (1) a

key of period 1 is chosen, or (2) a key of period 2 is chosen and both characters of the key are equal. The former occurs with probability $\frac{1}{2}$, and the latter occurs with probability $\frac{1}{2} \cdot \frac{1}{26}$. So

$$\Pr[\mathcal{A} \text{ outputs } 0 \mid b = 0] = \frac{1}{2} + \frac{1}{2} \cdot \frac{1}{26} \approx 0.52.$$

When $b = 1$ then $c_1 = c_2$ only if a key of period 2 is chosen and the first character of the key is one more than the second character of the key, which happens with probability $\frac{1}{2} \cdot \frac{1}{26}$. So

$$\Pr[\mathcal{A} \text{ outputs } 1 \mid b = 1] = 1 - \Pr[\mathcal{A} \text{ outputs } 0 \mid b = 1] = 1 - \frac{1}{2} \cdot \frac{1}{26} \approx 0.98.$$

Plugging into Equation (2.4) then gives

$$\Pr\left[\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi} = 1\right] = \frac{1}{2} \cdot \left(\frac{1}{2} + \frac{1}{2} \cdot \frac{1}{26} + 1 - \frac{1}{2} \cdot \frac{1}{26}\right) = 0.75 > \frac{1}{2},$$

and the scheme is not perfectly indistinguishable. ♢

2.2 The One-Time Pad

In 1917, Vernam patented a perfectly secret encryption scheme now called the *one-time pad*. At the time Vernam proposed the scheme, there was no proof that it was perfectly secret; in fact, the notion of perfect secrecy was not yet defined. Approximately 25 years later, however, Shannon introduced the definition of perfect secrecy and demonstrated that the one-time pad satisfied that definition.

In describing the scheme we let $a \oplus b$ denote the *bitwise exclusive-or* (XOR) of two equal-length binary strings a and b. (i.e., if $a = a_1 \cdots a_\ell$ and $b = b_1 \cdots b_\ell$ are ℓ-bit strings, then $a \oplus b$ is the ℓ-bit string given by $(a_1 \oplus b_1) \cdots (a_\ell \oplus b_\ell)$.) In the one-time pad encryption scheme the key is a uniform string of the same length as the message, and the ciphertext is computed by simply XORing the key and the message; a formal definition is given as Construction 2.9. Before discussing security, we first verify correctness: For every key k and every message m it holds that $\mathsf{Dec}_k(\mathsf{Enc}_k(m)) = k \oplus k \oplus m = m$, and so the one-time pad constitutes a valid encryption scheme.

One can easily prove perfect secrecy of the one-time pad using Lemma 2.5 because the ciphertext is uniformly distributed regardless of what message is encrypted. We give a direct proof based on the original definition.

THEOREM 2.10 *The one-time pad encryption scheme is perfectly secret.*

CONSTRUCTION 2.9

Fix an integer $\ell > 0$. The message space $\mathcal{M}$, key space $\mathcal{K}$, and ciphertext space $\mathcal{C}$ are all equal to $\{0,1\}^\ell$ (the set of all binary strings of length ℓ).

- Gen: the key-generation algorithm chooses a key from $\mathcal{K} = \{0,1\}^\ell$ according to the uniform distribution (i.e., each of the 2^ℓ strings in the space is chosen as the key with probability exactly $2^{-\ell}$).
- Enc: given a key $k \in \{0,1\}^\ell$ and a message $m \in \{0,1\}^\ell$, the encryption algorithm outputs the ciphertext $c := k \oplus m$.
- Dec: given a key $k \in \{0,1\}^\ell$ and a ciphertext $c \in \{0,1\}^\ell$, the decryption algorithm outputs the message $m := k \oplus c$.

The one-time pad encryption scheme.

PROOF We first compute $\Pr[C = c \mid M = m]$ for arbitrary $c \in \mathcal{C}$ and $m \in \mathcal{M}$ with $\Pr[M = m] > 0$. For the one-time pad, we have

$$\begin{aligned}\Pr[C = c \mid M = m] &= \Pr[K \oplus m = c \mid M = m] \\ &= \Pr[K = m \oplus c \mid M = m] \\ &= 2^{-\ell},\end{aligned}$$

where the first equality is by definition of the scheme and the fact that we condition on the event $M = m$, and the final equality holds because the key K is a uniform ℓ-bit string that is independent of M. Fix any distribution over $\mathcal{M}$. Using the above result, we see that for any $c \in \mathcal{C}$ we have

$$\begin{aligned}\Pr[C = c] &= \sum_{m \in \mathcal{M}} \Pr[C = c \mid M = m] \cdot \Pr[M = m] \\ &= 2^{-\ell} \cdot \sum_{m \in \mathcal{M}} \Pr[M = m] \\ &= 2^{-\ell},\end{aligned}$$

where the sum is over $m \in \mathcal{M}$ with $\Pr[M = m] \neq 0$. Bayes' Theorem gives:

$$\begin{aligned}\Pr[M = m \mid C = c] &= \frac{\Pr[C = c \mid M = m] \cdot \Pr[M = m]}{\Pr[C = c]} \\ &= \frac{2^{-\ell} \cdot \Pr[M = m]}{2^{-\ell}} \\ &= \Pr[M = m].\end{aligned}$$

We conclude that the one-time pad is perfectly secret. ■

The one-time pad was used by several national-intelligence agencies in the mid-20th century to encrypt sensitive traffic. Perhaps most famously, the "red phone" linking the White House and the Kremlin during the Cold War

was protected using one-time pad encryption, where the governments of the US and the USSR would exchange extremely long keys using trusted couriers carrying briefcases of paper on which random characters were written.

Notwithstanding the above, one-time pad encryption is rarely used nowadays because it has a number of drawbacks. Most prominent is that *the key is as long as the message*.[2] This limits the usefulness of the scheme for sending very long messages (as it may be difficult to securely share and store a very long key), and is problematic when the parties cannot predict in advance (an upper bound on) how long the message will be.

Moreover, the one-time pad—as the name indicates—*is only secure if used once* (with a given key). Although we did not yet define a notion of secrecy when multiple messages are encrypted, it is easy to see that encrypting more than one message with the same key leaks a lot of information. In particular, say two messages m, m' are encrypted using the same (unknown) key k. An adversary who obtains $c = m \oplus k$ and $c' = m' \oplus k$ can compute

$$c \oplus c' = (m \oplus k) \oplus (m' \oplus k) = m \oplus m'$$

and thus learn the XOR of the two messages or, equivalently, exactly where the two messages differ. This attack extends to more than two messages as well, where it enables the attacker to learn the XOR of all pairs of messages. While this may not seem very significant, it is enough to rule out any claims of perfect secrecy for encrypting more than one message using the same key. Moreover, if the messages correspond to natural-language text, then given the XOR of sufficiently many pairs of messages—or even two sufficiently long messages—it is possible to perform frequency analysis (as in the previous chapter, though more complex) and recover the messages themselves. (See Exercise 2.16 for an example.) An interesting historical example of this is given by the *VENONA project*, as part of which the US and UK were able to decrypt ciphertexts sent by the Soviet Union that were mistakenly encrypted with repeated portions of a one-time pad over several decades.

2.3 Limitations of Perfect Secrecy

We ended the previous section by noting some drawbacks of the one-time pad encryption scheme. Here, we show that these drawbacks are not specific to that scheme, but are instead *inherent* limitations of perfect secrecy. Specifically, we prove that *any* perfectly secret encryption scheme must have a key space that is at least as large as the message space. If all keys are the same

[2] This does not make the one-time pad useless, since it may be easier for two parties to share a key at some point in time before the message to be communicated is known.

length, and the message space consists of all strings of some fixed length, this implies that the key is at least as long as the message. In particular, the key length of the one-time pad is optimal. (The other limitation—namely, that a key can be used only once—is also inherent; see Exercise 2.19.)

THEOREM 2.11 *If* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *is a perfectly secret encryption scheme with message space* $\mathcal{M}$ *and key space* $\mathcal{K}$*, then* $|\mathcal{K}| \geq |\mathcal{M}|$*.*

PROOF We show that if $|\mathcal{K}| < |\mathcal{M}|$ then the scheme cannot be perfectly secret. Assume $|\mathcal{K}| < |\mathcal{M}|$. Consider the uniform distribution over $\mathcal{M}$ and let $c \in \mathcal{C}$ be a ciphertext that occurs with nonzero probability. Let $\mathcal{M}(c)$ be the set of all possible messages that are possible decryptions of c; that is

$$\mathcal{M}(c) \stackrel{\text{def}}{=} \{m \mid m = \mathsf{Dec}_k(c) \text{ for some } k \in \mathcal{K}\}.$$

Clearly $|\mathcal{M}(c)| \leq |\mathcal{K}|$. (Recall that we may assume Dec is deterministic.) If $|\mathcal{K}| < |\mathcal{M}|$, there is some $m' \in \mathcal{M}$ such that $m' \notin \mathcal{M}(c)$. But then

$$\Pr[M = m' \mid C = c] = 0 \neq \Pr[M = m'],$$

and so the scheme is not perfectly secret. ■

Perfect secrecy with shorter keys? The above theorem shows an inherent limitation of schemes that achieve perfect secrecy. Even so, individuals occasionally claim they have developed a radically new encryption scheme that is "unbreakable" and achieves the security of the one-time pad without using keys as long as what is being encrypted. The above proof demonstrates that such claims *cannot* be true; anyone making such claims either knows very little about cryptography or is blatantly lying.

2.4 *Shannon's Theorem

In his work on perfect secrecy, Shannon also provided a characterization of perfectly secret encryption schemes. This characterization says that, under certain conditions, the key-generation algorithm Gen must choose the key *uniformly* from the set of all possible keys (as in the one-time pad); moreover, for every message m and ciphertext c there is a *unique* key mapping m to c (again, as in the one-time pad). Beyond being interesting in its own right, this theorem is a useful tool for proving (or disproving) perfect secrecy of schemes. We discuss this further after the proof.

The theorem as stated here assumes $|\mathcal{M}| = |\mathcal{K}| = |\mathcal{C}|$, meaning that the sets of plaintexts, keys, and ciphertexts all have the same size. We have already

seen that for perfect secrecy we must have $|\mathcal{K}| \geq |\mathcal{M}|$. It is easy to see that correct decryption requires $|\mathcal{C}| \geq |\mathcal{M}|$. Therefore, in some sense, perfectly secret encryption schemes with $|\mathcal{M}| = |\mathcal{K}| = |\mathcal{C}|$ are "optimal."

THEOREM 2.12 (Shannon's theorem) *Let* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *be an encryption scheme with message space* $\mathcal{M}$*, for which* $|\mathcal{M}| = |\mathcal{K}| = |\mathcal{C}|$*. The scheme is perfectly secret if and only if:*

1. *Every key* $k \in \mathcal{K}$ *is chosen with (equal) probability* $1/|\mathcal{K}|$ *by* Gen*.*
2. *For every* $m \in \mathcal{M}$ *and every* $c \in \mathcal{C}$*, there is a unique key* $k \in \mathcal{K}$ *such that* $\mathsf{Enc}_k(m)$ *outputs* c*.*

PROOF The intuition behind the proof is as follows. To see that the stated conditions imply perfect secrecy, note that condition 2 means that any ciphertext c could be the result of encrypting any possible plaintext m, because there is some key k mapping m to c. Since there is a *unique* such key, and each key is chosen with equal probability, perfect secrecy follows as for the one-time pad. For the other direction, perfect secrecy immediately implies that for every m and c there is at least one key mapping m to c. The fact that $|\mathcal{M}| = |\mathcal{K}| = |\mathcal{C}|$ means, moreover, that for every m and c there is exactly *one* such key. Given this, each key must be chosen with equal probability or else perfect secrecy would fail to hold. A formal proof follows.

We assume for simplicity that Enc is deterministic. (One can show that this is without loss of generality here.) We first prove that if the encryption scheme satisfies conditions 1 and 2, then it is perfectly secret. The proof is essentially the same as the proof of perfect secrecy for the one-time pad, so we will be relatively brief. Fix arbitrary $c \in \mathcal{C}$ and $m \in \mathcal{M}$. Let k be the unique key, guaranteed by condition 2, for which $\mathsf{Enc}_k(m) = c$. Then,

$$\Pr[\mathsf{Enc}_K(m) = c] = \Pr[K = k] = 1/|\mathcal{K}|\,,$$

where the final equality holds by condition 1. Since this holds for arbitrary m and c, Lemma 2.5 implies that the scheme is perfectly secret.

For the second direction, assume the encryption scheme is perfectly secret; we show that conditions 1 and 2 hold. Fix arbitrary $c \in \mathcal{C}$. There must be some message m^* for which $\Pr[\mathsf{Enc}_K(m^*) = c] \neq 0$. Lemma 2.5 then implies that $\Pr[\mathsf{Enc}_K(m) = c] \neq 0$ for every $m \in \mathcal{M}$. In other words, if we let $\mathcal{M} = \{m_1, m_2, \ldots\}$, then for each $m_i \in \mathcal{M}$ we have a nonempty set of keys $\mathcal{K}_i \subset \mathcal{K}$ such that $\mathsf{Enc}_k(m_i) = c$ if and only if $k \in \mathcal{K}_i$. Moreover, when $i \neq j$ then $\mathcal{K}_i$ and $\mathcal{K}_j$ must be disjoint or else correctness fails to hold. Since $|\mathcal{K}| = |\mathcal{M}|$, we see that each $\mathcal{K}_i$ contains only a single key k_i, as required by condition 2. Now, Lemma 2.5 shows that for any $m_i, m_j \in \mathcal{M}$ we have

$$\Pr[K = k_i] = \Pr[\mathsf{Enc}_K(m_i) = c] = \Pr[\mathsf{Enc}_K(m_j) = c] = \Pr[K = k_j].$$

Since this holds for all $1 \leq i, j \leq |\mathcal{M}| = |\mathcal{K}|$, and $k_i \neq k_j$ for $i \neq j$, this means each key is chosen with probability $1/|\mathcal{K}|$, as required by condition 1. ■

Shannon's theorem is useful for deciding whether a given scheme is perfectly secret. Condition 1 is easy to check, and condition 2 can be demonstrated (or contradicted) without having to compute any probabilities (in contrast to working with Definition 2.3 directly). As an example, perfect secrecy of the one-time pad is trivial to prove using Shannon's theorem. We stress, however, that the theorem only applies when $|\mathcal{M}| = |\mathcal{K}| = |\mathcal{C}|$.

References and Additional Reading

The one-time pad is popularly credited to Vernam [200], who filed a patent on it, but recent historical research [28] shows that it was invented some 35 years earlier. Analysis of the one-time pad had to await the groundbreaking work of Shannon [177], who introduced the notion of perfect secrecy.

In this chapter we studied perfectly secret *encryption*. Some other cryptographic problems can also be solved with "perfect" security. A notable example is the problem of message authentication where the aim is to prevent an adversary from (undetectably) modifying a message sent from one party to another. We study this problem in depth in Chapter 4, discussing "perfectly secure" message authentication in Section 4.6.

Exercises

2.1 Prove that, by redefining the key space, we may assume that the key-generation algorithm Gen chooses a uniform key from the key space, without changing $\Pr[C = c \mid M = m]$ for any m, c.

> **Hint:** Define the key space to be the set of all possible random bits used by the randomized algorithm Gen.

2.2 Prove that, by redefining the key space as well as the encryption algorithm, we may assume that encryption is deterministic without changing $\Pr[C = c \mid M = m]$ for any m, c.

2.3 Prove or refute: An encryption scheme with message space $\mathcal{M}$ is perfectly secret if and only if for every probability distribution on $\mathcal{M}$ and every $c_0, c_1 \in \mathcal{C}$ we have $\Pr[C = c_0] = \Pr[C = c_1]$.

2.4 Prove or refute: For every perfectly secret encryption scheme it holds that for every distribution on the message space $\mathcal{M}$, every $m, m' \in \mathcal{M}$, and every $c \in \mathcal{C}$:

$$\Pr[M = m \mid C = c] = \Pr[M = m' \mid C = c].$$

2.5 Prove that in Definition 2.6 we may assume $\mathcal{A}$ is deterministic without loss of generality.

2.6 Prove Lemma 2.7.

2.7 What is the ciphertext that results when the plaintext `0x012345` (written in hex) is encrypted using the one-time pad with key `0xFFEEDD`?

2.8 For each of the following encryption schemes, state whether the scheme is perfectly secret. Justify your answer in each case.

(a) The message space is $\mathcal{M} = \{0, \ldots, 4\}$, and Gen chooses a uniform key from the key space $\mathcal{K} = \{0, \ldots, 5\}$. $\mathsf{Enc}_k(m)$ returns $[m + k \bmod 5]$, and $\mathsf{Dec}_k(c)$ returns $[c - k \bmod 5]$.

(b) The message space is $\mathcal{M} = \{m \in \{0,1\}^\ell \mid \text{the last bit of } m \text{ is } 0\}$. Gen chooses a uniform key from $\{0,1\}^{\ell-1}$. $\mathsf{Enc}_k(m)$ returns ciphertext $m \oplus (k \| 0)$, and $\mathsf{Dec}_k(c)$ returns $c \oplus (k \| 0)$.

2.9 In each of the following schemes, $\mathsf{Enc}_k(m) = [m+k \bmod 3]$. State in each case whether the scheme is perfectly secret, and justify your answers.

(a) The message space is $\mathcal{M} = \{0,1\}$, and Gen chooses a uniform key from the key space $\mathcal{K} = \{0,1\}$.

(b) The message space is $\mathcal{M} = \{0,1,2\}$, and Gen chooses a uniform key from the key space $\mathcal{K} = \{0,1,2\}$.

(c) The message space is $\mathcal{M} = \{0,1\}$, and Gen chooses a uniform key from the key space $\mathcal{K} = \{0,1,2\}$.

2.10 The following questions concern the message space $\mathcal{M} = \{0,1\}^{\leq \ell}$, the set of all nonempty binary strings of length at most ℓ.

(a) Consider the encryption scheme in which Gen chooses a uniform key from $\mathcal{K} = \{0,1\}^\ell$, and $\mathsf{Enc}_k(m)$ outputs $k_{|m|} \oplus m$, where k_t denotes the first t bits of k. Show that this scheme is not perfectly secret for message space $\mathcal{M}$.

(b) Design a perfectly secret encryption scheme for message space $\mathcal{M}$.

2.11 When using the one-time pad with the key $k = 0^\ell$, we have $\mathsf{Enc}_k(m) = k \oplus m = m$ and the message is sent in the clear! It has therefore been suggested to modify the one-time pad by only encrypting with $k \neq 0^\ell$ (i.e., to have Gen choose k uniformly from the set of *nonzero* keys of length ℓ). Is this modified scheme still perfectly secret? Explain.

2.12 Let Π denote the Vigenère cipher where the message space consists of all 3-character strings (over the English alphabet), and the period t is fixed to 2 (and so the key is a uniform string of length 2). Define $\mathcal{A}$ as follows: $\mathcal{A}$ outputs $m_0 = \texttt{aaa}$ and $m_1 = \texttt{aab}$. When given a ciphertext c, it outputs 0 if the first character of c is the same as the third character of c, and outputs 1 otherwise. Compute $\Pr[\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi} = 1]$.

2.13 Let Π denote the Vigenère cipher where the message space consists of all 3-character strings (over the English alphabet), and the key is generated by first choosing the period t uniformly from $\{1, 2, 3\}$ and then letting the key be a uniform string of length t.

(a) Define $\mathcal{A}$ as follows: $\mathcal{A}$ outputs $m_0 = \texttt{aab}$ and $m_1 = \texttt{abb}$. When given a ciphertext c, it outputs 0 if the first character of c is the same as the second character of c, and outputs 1 otherwise. Compute $\Pr[\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi} = 1]$.

(b) Construct and analyze an adversary $\mathcal{A}'$ for which $\Pr[\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A}',\Pi} = 1]$ is greater than your answer from part (a).

2.14 In this exercise, we look at different conditions under which the shift, mono-alphabetic substitution, and Vigenère ciphers are perfectly secret:

(a) Prove that if only a single character is encrypted, then the shift cipher is perfectly secret.

(b) What is the largest message space $\mathcal{M}$ for which the mono-alphabetic substitution cipher provides perfect secrecy?

(c) Prove that the Vigenère cipher using (fixed) period t is perfectly secret when used to encrypt messages of length t.

Reconcile this with the attacks shown in the previous chapter.

2.15 Give a direct proof that a scheme satisfying Definition 2.6 must have $|\mathcal{K}| \geq |\mathcal{M}|$. Specifically, let Π be an arbitrary encryption scheme with $|\mathcal{K}| < |\mathcal{M}|$. Show an $\mathcal{A}$ for which $\Pr\left[\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi} = 1\right] > \frac{1}{2}$.

Hint: It may be easier to let $\mathcal{A}$ be randomized.

2.16 The following questions concern multiple encryptions of single-character ASCII plaintexts with the one-time pad using the same 8-bit key. You may assume that the plaintexts are either (upper- or lower-case) English letters or the space character.

(a) Say you see the ciphertexts 1011 0111 and 1110 0111. What can you deduce about the plaintext characters these correspond to?

(b) Say you see the three ciphertexts 0110 0110, 0011 0010, and 0010 0011. What can you deduce about the plaintext characters these correspond to?

Hint: Focus on the second bit of the ciphertexts.

2.17 Assume we require only that an encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ with message space $\mathcal{M}$ satisfy the following: For all $m \in \mathcal{M}$, we have $\Pr[\mathsf{Dec}_K(\mathsf{Enc}_K(m)) = m] \geq 2^{-t}$. (This probability is taken over choice of the key as well as any randomness used during encryption/decryption.) Show that perfect secrecy can be achieved with $|\mathcal{K}| < |\mathcal{M}|$ when $t \geq 1$. Prove a lower bound on the size of $\mathcal{K}$ in terms of t.

2.18 Let $\varepsilon > 0$ be a constant. Say an encryption scheme is *ε-perfectly secret* if for every adversary $\mathcal{A}$ it holds that

$$\Pr\left[\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi} = 1\right] \leq \frac{1}{2} + \varepsilon\,.$$

(Compare to Definition 2.6.) Consider a variant of the one-time pad where $\mathcal{M} = \{0,1\}^\ell$ and the key is chosen uniformly from an arbitrary set $\mathcal{K} \subseteq \{0,1\}^\ell$ with $|\mathcal{K}| = (1-\varepsilon)\cdot 2^\ell$; encryption and decryption are otherwise the same.

(a) Prove that this scheme is ε-perfectly secret.

(b) Prove that this scheme is $\left(\frac{\varepsilon}{2(1-\varepsilon)}\right)$-perfectly secret when $\varepsilon \leq 1/2$. (Note that $\frac{\varepsilon}{2(1-\varepsilon)} \leq \varepsilon$ here, so this is an improvement over part (a).)

(c) Prove that any deterministic scheme that is ε-perfectly secret must have $|\mathcal{K}| \geq (1-2\varepsilon)\cdot|\mathcal{M}|$. (Note: It is an open question to prove a tight lower bound that also holds for randomized schemes.)

2.19 In this problem we consider definitions of perfect secrecy for the encryption of *two* messages (using the same key). Here we consider distributions on *pairs* of messages from the message space $\mathcal{M}$; we let M_1, M_2 be random variables denoting the first and second message, respectively. (These random variables are not assumed to be independent.) We generate a (single) key k, sample a pair of messages (m_1, m_2) according to the given distribution, and then compute ciphertexts $c_1 \leftarrow \mathsf{Enc}_k(m_1)$ and $c_2 \leftarrow \mathsf{Enc}_k(m_2)$; this induces a distribution on pairs of ciphertexts and we let C_1, C_2 be the corresponding random variables.

(a) Say encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is *perfectly secret for two messages* if for all distributions on $\mathcal{M}\times\mathcal{M}$, all $m_1, m_2 \in \mathcal{M}$, and all ciphertexts $c_1, c_2 \in \mathcal{C}$ with $\Pr[C_1 = c_1 \wedge C_2 = c_2] > 0$:

$$\begin{aligned}\Pr\left[M_1 = m_1 \wedge M_2 = m_2 \mid C_1 = c_1 \wedge C_2 = c_2\right]\\ = \Pr[M_1 = m_1 \wedge M_2 = m_2].\end{aligned}$$

Prove that *no* encryption scheme can satisfy this definition.

Hint: Take $c_1 = c_2$.

(b) Say encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is *perfectly secret for two distinct messages* if for all distributions on $\mathcal{M} \times \mathcal{M}$ where the first and second messages are guaranteed to be different (i.e., distributions on pairs of *distinct* messages), all $m_1, m_2 \in \mathcal{M}$, and all $c_1, c_2 \in \mathcal{C}$ with $\Pr[C_1 = c_1 \wedge C_2 = c_2] > 0$:

$$\Pr[M_1 = m_1 \wedge M_2 = m_2 \mid C_1 = c_1 \wedge C_2 = c_2] = \Pr[M_1 = m_1 \wedge M_2 = m_2].$$

Show an encryption scheme that provably satisfies this definition.

Hint: The encryption scheme you propose need not be efficient, although an efficient solution is possible.

Part II

Private-Key (Symmetric) Cryptography