TEXTBOOKS IN MATHEMATICS

# Cryptography

## Theory and Practice

FOURTH EDITION

Douglas R. Stinson
Maura B. Paterson

CRC Press
Taylor & Francis Group
A CHAPMAN & HALL BOOK

# Cryptography

## Theory and Practice

Fourth Edition

# Textbooks in Mathematics

## Series editors:
## Al Boggess and Ken Rosen

**MATHEMATICAL MODELING FOR BUSINESS ANALYTICS**
*William P. Fox*

**ELEMENTARY LINEAR ALGEBRA**
*James R. Kirkwood and Bessie H. Kirkwood*

**APPLIED FUNCTIONAL ANALYSIS, THIRD EDITION**
*J. Tinsley Oden and Leszek Demkowicz*

**AN INTRODUCTION TO NUMBER THEORY WITH CRYPTOGRAPHY, SECOND EDITION**
*James R. Kraft and Lawrence Washington*

**MATHEMATICAL MODELING: BRANCHING BEYOND CALCULUS**
*Crista Arangala, Nicolas S. Luke and Karen A. Yokley*

**ELEMENTARY DIFFERENTIAL EQUATIONS, SECOND EDITION**
*Charles Roberts*

**ELEMENTARY INTRODUCTION TO THE LEBESGUE INTEGRAL**
*Steven G. Krantz*

**LINEAR METHODS FOR THE LIBERAL ARTS**
*David Hecker and Stephen Andrilli*

**CRYPTOGRAPHY: THEORY AND PRACTICE, FOURTH EDITION**
*Douglas R. Stinson and Maura B. Paterson*

**DISCRETE MATHEMATICS WITH DUCKS, SECOND EDITION**
*Sarah-Marie Belcastro*

**BUSINESS PROCESS MODELING, SIMULATION AND DESIGN, THIRD EDITION**
*Manual Laguna and Johan Marklund*

**GRAPH THEORY AND ITS APPLICATIONS, THIRD EDITION**
*Jonathan L. Gross, Jay Yellen and Mark Anderson*

# Cryptography

## Theory and Practice

Fourth Edition

Douglas R. Stinson
Maura B. Paterson


CRC Press
Taylor & Francis Group
Boca Raton London New York

CRC Press is an imprint of the
Taylor & Francis Group, an **informa** business

CRC Press
Taylor & Francis Group
6000 Broken Sound Parkway NW, Suite 300
Boca Raton, FL 33487-2742



Printed on acid-free paper
Version Date: 20180724

International Standard Book Number-13: 978-1-1381-9701-5 (Hardback)



**Library of Congress Cataloging-in-Publication Data**

Names: Stinson, Douglas R. (Douglas Robert), 1956- author. | Paterson, Maura B., author.
Title: Cryptography : theory and practice / Douglas R. Stinson and Maura B. Paterson.
Description: Fourth edition. | Boca Raton : CRC Press, Taylor & Francis Group, 2018.
Identifiers: LCCN 2018018724 | ISBN 9781138197015
Subjects: LCSH: Coding theory. | Cryptography.
Classification: LCC QA268 .S75 2018 | DDC 005.8/2--dc23
LC record available at https://lccn.loc.gov/2018018724

**Visit the Taylor & Francis Web site at**
**http://www.taylorandfrancis.com**

**and the CRC Press Web site at**
**http://www.crcpress.com**

*To my children, Michela and Aiden*
*DRS*

*To my father, Hamish*
*MBP*

# *Contents*

**Preface** **xv**

**1 Introduction to Cryptography** **1**
1.1 Cryptosystems and Basic Cryptographic Tools . . . . . 1
1.1.1 Secret-key Cryptosystems . . . . . 1
1.1.2 Public-key Cryptosystems . . . . . 2
1.1.3 Block and Stream Ciphers . . . . . 3
1.1.4 Hybrid Cryptography . . . . . 3
1.2 Message Integrity . . . . . 4
1.2.1 Message Authentication Codes . . . . . 6
1.2.2 Signature Schemes . . . . . 6
1.2.3 Nonrepudiation . . . . . 7
1.2.4 Certificates . . . . . 8
1.2.5 Hash Functions . . . . . 8
1.3 Cryptographic Protocols . . . . . 9
1.4 Security . . . . . 10
1.5 Notes and References . . . . . 13

**2 Classical Cryptography** **15**
2.1 Introduction: Some Simple Cryptosystems . . . . . 15
2.1.1 The Shift Cipher . . . . . 17
2.1.2 The Substitution Cipher . . . . . 20
2.1.3 The Affine Cipher . . . . . 22
2.1.4 The Vigenère Cipher . . . . . 26
2.1.5 The Hill Cipher . . . . . 27
2.1.6 The Permutation Cipher . . . . . 32
2.1.7 Stream Ciphers . . . . . 34
2.2 Cryptanalysis . . . . . 38
2.2.1 Cryptanalysis of the Affine Cipher . . . . . 40
2.2.2 Cryptanalysis of the Substitution Cipher . . . . . 42
2.2.3 Cryptanalysis of the Vigenère Cipher . . . . . 45
2.2.4 Cryptanalysis of the Hill Cipher . . . . . 48
2.2.5 Cryptanalysis of the LFSR Stream Cipher . . . . . 49
2.3 Notes and References . . . . . 51
Exercises . . . . . 51

**3 Shannon's Theory, Perfect Secrecy, and the One-Time Pad** **61**
3.1 Introduction . . . 61
3.2 Elementary Probability Theory . . . 62
3.3 Perfect Secrecy . . . 64
3.4 Entropy . . . 70
3.4.1 Properties of Entropy . . . 72
3.5 Spurious Keys and Unicity Distance . . . 75
3.6 Notes and References . . . 79
Exercises . . . 80

**4 Block Ciphers and Stream Ciphers** **83**
4.1 Introduction . . . 83
4.2 Substitution-Permutation Networks . . . 84
4.3 Linear Cryptanalysis . . . 89
4.3.1 The Piling-up Lemma . . . 89
4.3.2 Linear Approximations of S-boxes . . . 91
4.3.3 A Linear Attack on an SPN . . . 94
4.4 Differential Cryptanalysis . . . 98
4.5 The Data Encryption Standard . . . 105
4.5.1 Description of DES . . . 105
4.5.2 Analysis of DES . . . 107
4.6 The Advanced Encryption Standard . . . 109
4.6.1 Description of AES . . . 110
4.6.2 Analysis of AES . . . 115
4.7 Modes of Operation . . . 116
4.7.1 Padding Oracle Attack on CBC Mode . . . 120
4.8 Stream Ciphers . . . 122
4.8.1 Correlation Attack on a Combination Generator . . . 123
4.8.2 Algebraic Attack on a Filter Generator . . . 127
4.8.3 Trivium . . . 130
4.9 Notes and References . . . 131
Exercises . . . 131

**5 Hash Functions and Message Authentication** **137**
5.1 Hash Functions and Data Integrity . . . 137
5.2 Security of Hash Functions . . . 139
5.2.1 The Random Oracle Model . . . 140
5.2.2 Algorithms in the Random Oracle Model . . . 142
5.2.3 Comparison of Security Criteria . . . 146
5.3 Iterated Hash Functions . . . 148
5.3.1 The Merkle-Damgård Construction . . . 151
5.3.2 Some Examples of Iterated Hash Functions . . . 156
5.4 The Sponge Construction . . . 157
5.4.1 SHA-3 . . . 160
5.5 Message Authentication Codes . . . 161

5.5.1 Nested MACs and HMAC . . . 163
5.5.2 CBC-MAC . . . 166
5.5.3 Authenticated Encryption . . . 167
5.6 Unconditionally Secure MACs . . . 170
5.6.1 Strongly Universal Hash Families . . . 173
5.6.2 Optimality of Deception Probabilities . . . 175
5.7 Notes and References . . . 177
Exercises . . . 178

**6 The RSA Cryptosystem and Factoring Integers 185**
6.1 Introduction to Public-key Cryptography . . . 185
6.2 More Number Theory . . . 188
6.2.1 The Euclidean Algorithm . . . 188
6.2.2 The Chinese Remainder Theorem . . . 191
6.2.3 Other Useful Facts . . . 194
6.3 The RSA Cryptosystem . . . 196
6.3.1 Implementing RSA . . . 198
6.4 Primality Testing . . . 200
6.4.1 Legendre and Jacobi Symbols . . . 202
6.4.2 The Solovay-Strassen Algorithm . . . 205
6.4.3 The Miller-Rabin Algorithm . . . 208
6.5 Square Roots Modulo $n$ . . . 210
6.6 Factoring Algorithms . . . 211
6.6.1 The Pollard $p-1$ Algorithm . . . 212
6.6.2 The Pollard Rho Algorithm . . . 213
6.6.3 Dixon's Random Squares Algorithm . . . 216
6.6.4 Factoring Algorithms in Practice . . . 221
6.7 Other Attacks on RSA . . . 223
6.7.1 Computing $\phi(n)$ . . . 223
6.7.2 The Decryption Exponent . . . 223
6.7.3 Wiener's Low Decryption Exponent Attack . . . 228
6.8 The Rabin Cryptosystem . . . 232
6.8.1 Security of the Rabin Cryptosystem . . . 234
6.9 Semantic Security of RSA . . . 236
6.9.1 Partial Information Concerning Plaintext Bits . . . 237
6.9.2 Obtaining Semantic Security . . . 239
6.10 Notes and References . . . 245
Exercises . . . 246

**7 Public-Key Cryptography and Discrete Logarithms 255**
7.1 Introduction . . . 255
7.1.1 The ElGamal Cryptosystem . . . 256
7.2 Algorithms for the Discrete Logarithm Problem . . . 258
7.2.1 Shanks' Algorithm . . . 258
7.2.2 The Pollard Rho Discrete Logarithm Algorithm . . . 260

7.2.3 The Pohlig-Hellman Algorithm . . . 263
7.2.4 The Index Calculus Method . . . 266
7.3 Lower Bounds on the Complexity of Generic Algorithms . 268
7.4 Finite Fields . . . 272
7.4.1 Joux's Index Calculus . . . 276
7.5 Elliptic Curves . . . 278
7.5.1 Elliptic Curves over the Reals . . . 278
7.5.2 Elliptic Curves Modulo a Prime . . . 281
7.5.3 Elliptic Curves over Finite Fields . . . 284
7.5.4 Properties of Elliptic Curves . . . 285
7.5.5 Pairings on Elliptic Curves . . . 286
7.5.6 ElGamal Cryptosystems on Elliptic Curves . . . 290
7.5.7 Computing Point Multiples on Elliptic Curves . . . 292
7.6 Discrete Logarithm Algorithms in Practice . . . 294
7.7 Security of ElGamal Systems . . . 296
7.7.1 Bit Security of Discrete Logarithms . . . 296
7.7.2 Semantic Security of ElGamal Systems . . . 299
7.7.3 The Diffie-Hellman Problems . . . 300
7.8 Notes and References . . . 301
Exercises . . . 302

**8 Signature Schemes 309**
8.1 Introduction . . . 309
8.1.1 RSA Signature Scheme . . . 310
8.2 Security Requirements for Signature Schemes . . . 312
8.2.1 Signatures and Hash Functions . . . 313
8.3 The ElGamal Signature Scheme . . . 314
8.3.1 Security of the ElGamal Signature Scheme . . . 317
8.4 Variants of the ElGamal Signature Scheme . . . 320
8.4.1 The Schnorr Signature Scheme . . . 320
8.4.2 The Digital Signature Algorithm . . . 322
8.4.3 The Elliptic Curve DSA . . . 325
8.5 Full Domain Hash . . . 326
8.6 Certificates . . . 330
8.7 Signing and Encrypting . . . 331
8.8 Notes and References . . . 333
Exercises . . . 334

**9 Post-Quantum Cryptography 341**
9.1 Introduction . . . 341
9.2 Lattice-based Cryptography . . . 344
9.2.1 NTRU . . . 344
9.2.2 Lattices and the Security of NTRU . . . 348
9.2.3 Learning With Errors . . . 351
9.3 Code-based Cryptography and the McEliece Cryptosystem 353

9.4 Multivariate Cryptography . . . 358
9.4.1 Hidden Field Equations . . . 359
9.4.2 The Oil and Vinegar Signature Scheme . . . 364
9.5 Hash-based Signature Schemes . . . 367
9.5.1 Lamport Signature Scheme . . . 368
9.5.2 Winternitz Signature Scheme . . . 370
9.5.3 Merkle Signature Scheme . . . 373
9.6 Notes and References . . . 376
Exercises . . . 376

**10 Identification Schemes and Entity Authentication 379**
10.1 Introduction . . . 379
10.1.1 Passwords . . . 381
10.1.2 Secure Identification Schemes . . . 383
10.2 Challenge-and-Response in the Secret-key Setting . . . 384
10.2.1 Attack Model and Adversarial Goals . . . 389
10.2.2 Mutual Authentication . . . 391
10.3 Challenge-and-Response in the Public-key Setting . . . 394
10.3.1 Public-key Identification Schemes . . . 394
10.4 The Schnorr Identification Scheme . . . 397
10.4.1 Security of the Schnorr Identification Scheme . . . 400
10.5 The Feige-Fiat-Shamir Identification Scheme . . . 406
10.6 Notes and References . . . 411
Exercises . . . 412

**11 Key Distribution 415**
11.1 Introduction . . . 415
11.1.1 Attack Models and Adversarial Goals . . . 418
11.2 Key Predistribution . . . 419
11.2.1 Diffie-Hellman Key Predistribution . . . 419
11.2.2 The Blom Scheme . . . 421
11.2.3 Key Predistribution in Sensor Networks . . . 428
11.3 Session Key Distribution Schemes . . . 432
11.3.1 The Needham-Schroeder Scheme . . . 432
11.3.2 The Denning-Sacco Attack on the NS Scheme . . . 433
11.3.3 Kerberos . . . 435
11.3.4 The Bellare-Rogaway Scheme . . . 438
11.4 Re-keying and the Logical Key Hierarchy . . . 441
11.5 Threshold Schemes . . . 444
11.5.1 The Shamir Scheme . . . 445
11.5.2 A Simplified $(t, t)$-threshold Scheme . . . 448
11.5.3 Visual Threshold Schemes . . . 450
11.6 Notes and References . . . 454
Exercises . . . 454

**12 Key Agreement Schemes** **461**
12.1 Introduction . . . 461
12.1.1 Transport Layer Security (TLS) . . . 461
12.2 Diffie-Hellman Key Agreement . . . 463
12.2.1 The Station-to-station Key Agreement Scheme . . . 465
12.2.2 Security of STS . . . 466
12.2.3 Known Session Key Attacks . . . 469
12.3 Key Derivation Functions . . . 471
12.4 MTI Key Agreement Schemes . . . 472
12.4.1 Known Session Key Attacks on MTI/A0 . . . 476
12.5 Deniable Key Agreement Schemes . . . 478
12.6 Key Updating . . . 481
12.7 Conference Key Agreement Schemes . . . 484
12.8 Notes and References . . . 488
Exercises . . . 488

**13 Miscellaneous Topics** **491**
13.1 Identity-based Cryptography . . . 491
13.1.1 The Cocks Identity-based Cryptosystem . . . 492
13.1.2 Boneh-Franklin Identity-based Cryptosystem . . . 498
13.2 The Paillier Cryptosystem . . . 503
13.3 Copyright Protection . . . 506
13.3.1 Fingerprinting . . . 507
13.3.2 Identifiable Parent Property . . . 509
13.3.3 2-IPP Codes . . . 511
13.3.4 Tracing Illegally Redistributed Keys . . . 514
13.4 Bitcoin and Blockchain Technology . . . 518
13.5 Notes and References . . . 522
Exercises . . . 523

**A Number Theory and Algebraic Concepts for Cryptography** **527**
A.1 Modular Arithmetic . . . 527
A.2 Groups . . . 528
A.2.1 Orders of Group Elements . . . 530
A.2.2 Cyclic Groups and Primitive Elements . . . 531
A.2.3 Subgroups and Cosets . . . 532
A.2.4 Group Isomorphisms and Homomorphisms . . . 533
A.2.5 Quadratic Residues . . . 534
A.2.6 Euclidean Algorithm . . . 535
A.2.7 Direct Products . . . 536
A.3 Rings . . . 536
A.3.1 The Chinese Remainder Theorem . . . 538
A.3.2 Ideals and Quotient Rings . . . 539
A.4 Fields . . . 540

**B Pseudorandom Bit Generation for Cryptography** **543**
B.1 Bit Generators . . . . . 543
B.2 Security of Pseudorandom Bit Generators . . . . . 548
B.3 Notes and References . . . . . 550

**Bibliography** **551**

**Index** **567**

# *Preface*

The first edition of this book was published in 1995. The objective at that time was to produce a general textbook that treated all the essential core areas of cryptography, as well as a selection of more advanced topics. More recently, a second edition was published in 2002 and the third edition appeared in 2006.

There have been many exciting advances in cryptography since the publication of the first edition of this book 23 years ago. At the same time, many of the "core" areas of cryptography that were important then are still relevant now—providing a strong grounding in the fundamentals remains a primary goal of this book. Many decisions had to be made in terms of which older topics to retain and which new subjects should be incorporated into the book. Our choices were guided by criteria such as the relevance to practical applications of cryptography as well as the influence of new approaches and techniques to the design and analysis of cryptographic protocols. In many cases, this involved studying cutting-edge research and attempting to present it in an accessible manner suitable for presentation in the classroom.

In light of the above, the basic core material of secret-key and public-key cryptography is treated in a similar fashion as in previous editions. However, there are many topics that have been added to this edition, the most important being the following:

- There is a brand new chapter on the exciting, emerging area of post-quantum cryptography, which covers the most important cryptosystems that are designed to provide security against attacks by quantum computers (Chapter 9).
- A new high-level, nontechnical overview of the goals and tools of cryptography has been added (Chapter 1).
- A new mathematical appendix is included, which summarizes definitions and main results on number theory and algebra that are used throughout the book. This provides a quick way to reference any mathematical terms or theorems that a reader might wish to find (Appendix A).
- An expanded treatment of stream ciphers is provided, including common design techniques along with a description of the popular stream cipher known as *Trivium*.
- The book now presents additional interesting attacks on cryptosystems, including:

  - padding oracle attack
  - correlation attacks and algebraic attacks on stream ciphers
  - attack on the *DUAL-EC* random bit generator that makes use of a trapdoor.

- A treatment of the sponge construction for hash functions and its use in the new *SHA-3* hash standard is provided. This is a significant new approach to the design of hash functions.
- Methods of key distribution in sensor networks are described.
- There is a section on the basics of visual cryptography. This allows a secure method to split a secret visual message into pieces (shares) that can later be combined to reconstruct the secret.
- The fundamental techniques of cryptocurrencies, as used in BITCOIN and blockchain, are described.
- We explain the basics of the new cryptographic methods employed in messaging protocols such as *Signal*. This includes topics such as deniability and Diffie-Hellman key ratcheting.

We hope that this book can be used in a variety of courses. An introductory undergraduate level course could be based on a selection of material from the first eight chapters. We should point out that, in several chapters, the later sections can be considered to be more advanced than earlier sections. These sections could provide material for graduate courses or for self-study. Material in later chapters can also be included in an introductory or follow-up course, depending on the interests of the instructor.

Cryptography is a broad subject, and it requires knowledge of several areas of mathematics, including number theory, groups, rings and fields, linear algebra, probability and information theory. As well, some familiarity with computational complexity, algorithms, and NP-completeness theory is useful. In our opinion, it is the breadth of mathematical background required that often creates difficulty for students studying cryptography for the first time. With this in mind, we have maintained the mathematical presentation from previous editions. One basic guiding principle is that understanding relevant mathematics is essential to the comprehension of the various cryptographic schemes and topics. At the same time, we try to avoid unnecessarily advanced mathematical techniques—we provide the essentials, but we do not overload the reader with superfluous mathematical concepts.

The following features are common to all editions of this book:

- Mathematical background is provided where it is needed, in a "just-in-time" fashion.
- Informal descriptions of the cryptosystems are given along with more precise pseudo-code descriptions.

- Numerical examples are presented to illustrate the workings of most of the algorithms described in the book.
- The mathematical underpinnings of the algorithms and cryptosystems are explained carefully and rigorously.
- Numerous exercises are included, some of them quite challenging.

We have received useful feedback from various people on the content of this book as we prepared this new edition. In particular, we would like to thank Colleen Swanson for many helpful comments and suggestions. Several anonymous reviewers provided useful suggestions, and we also appreciate comments from Steven Galbraith and Jalaj Upadhyay. Finally, we thank Roberto De Prisco, who prepared the examples of shares in a visual threshold scheme that are included in Chapter 11.

**Douglas R. Stinson**
**Maura B. Paterson**

# Chapter 1

# Introduction to Cryptography

In this chapter, we present a brief overview of the kinds of problems studied in cryptography and the techniques used to solve them. These problems and the cryptographic tools that are employed in their solution are discussed in more detail and rigor in the rest of this book. This introduction may serve to provide an informal, non-technical, non-mathematical summary of the topics to be addressed. As such, it can be considered to be optional reading.

## 1.1 Cryptosystems and Basic Cryptographic Tools

In this section, we discuss basic notions relating to encryption. This includes secret-key and public-key cryptography, block and stream ciphers, and hybrid cryptography.

### 1.1.1 Secret-key Cryptosystems

Cryptography has been used for thousands of years to help to provide confidential communications between mutually trusted parties. In its most basic form, two people, often denoted as ***Alice*** and ***Bob***, have agreed on a particular ***secret key***. At some later time, Alice may wish to send a secret message to Bob (or Bob might want to send a message to Alice). The key is used to transform the original message (which is usually termed the ***plaintext***) into a scrambled form that is unintelligible to anyone who does not possess the key. This process is called ***encryption*** and the scrambled message is called the ***ciphertext***. When Bob receives the ciphertext, he can use the key to transform the ciphertext back into the original plaintext; this is the ***decryption*** process. A ***cryptosystem*** constitutes a complete specification of the keys and how they are used to encrypt and decrypt information.

Various types of cryptosystems of increasing sophistication have been used for many purposes throughout history. Important applications have included sensitive communications between political leaders and/or royalty, military maneuvers, etc. However, with the development of the internet and applications such as electronic commerce, many new diverse applications have emerged. These include scenarios such as encryption of passwords, credit card numbers, email, documents, files, and digital media.

It should also be mentioned that cryptographic techniques are also widely used to protect stored data in addition to data that is transmitted from one party to another. For example, users may wish to encrypt data stored on laptops, on external hard disks, in the cloud, in databases, etc. Additionally, it might be useful to be able to perform computations on encrypted data (without first decrypting the data).

The development and deployment of a cryptosystem must address the issue of security. Traditionally, the threat that cryptography addressed was that of an eavesdropping adversary who might intercept the ciphertext and attempt to decrypt it. If the adversary happens to possess the key, then there is nothing that can be done. Thus the main security consideration involves an adversary who does not possess the key, who is still trying to decrypt the ciphertext. The techniques used by the adversary to attempt to "break" the cryptosystem are termed ***cryptanalysis***. The most obvious type of cryptanalysis is to try to guess the key. An attack wherein the adversary tries to decrypt the ciphertext with every possible key in turn is termed an ***exhaustive key search***. When the adversary tries the correct key, the plaintext will be found, but when any other key is used, the "decrypted" ciphertext will likely be random gibberish. So an obvious first step in designing a secure cryptosystem is to specify a very large number of possible keys, so many that the adversary will not be able to test them all in any reasonable amount of time.

The model of cryptography described above is usually called ***secret-key cryptography***. This indicates that there is one secret key, which is known to both Alice and Bob. That is, the key is a "secret" that is known to two parties. This key is employed both to encrypt plaintexts and to decrypt ciphertexts. The actual encryption and decryption functions are thus inverses of each other. Some basic secret-key cryptosystems are introduced and analyzed with respect to different security notions in Chapters 2 and 3.

The drawback of secret-key cryptography is that Alice and Bob must somehow be able to agree on the secret key ahead of time (before they want to send any messages to each other). This might be straightforward if Alice and Bob are in the same place when they choose their secret key. But what if Alice and Bob are far apart, say on different continents? One possible solution is for Alice and Bob to use a public-key cryptosystem.

### 1.1.2 Public-key Cryptosystems

The revolutionary idea of ***public-key cryptography*** was introduced in the 1970s by Diffie and Hellman. Their idea was that it might be possible to devise a cryptosystem in which there are two distinct keys. A ***public key*** would be used to encrypt the plaintext and a ***private key*** would enable the ciphertext to be decrypted. Note that a public key can be known to "everyone," whereas a private key is known to only one person (namely, the recipient of the encrypted message). So a public-key cryptosystem would enable anyone to encrypt a message to be transmitted to Bob, say, and only Bob could decrypt the message. The first and best-known example of a public-key cryptosystem is the *RSA Cyptosystem* that

was invented by Rivest, Shamir and Adleman. Various types of public-key cryptosystems are presented in Chapters 6, 7, and 9.

Public-key cryptography obviates the need for two parties to agree on a prior shared secret key. However, it is still necessary to devise a method to distribute public keys securely. But this is not necessarily a trivial goal to accomplish, the main issue being the correctness or authenticity of purported public keys. Certificates, which we will discuss a bit later, are one common method to deal with this problem.

### 1.1.3 Block and Stream Ciphers

Cryptosystems are usually categorized as ***block ciphers*** or ***stream ciphers***. In a block cipher, the plaintext is divided into fixed-sized chunks called ***blocks***. A block is specified to be a bitstring (i.e., a string of 0's and 1's) of some fixed length (e.g., 64 or 128 bits). A block cipher will encrypt (or decrypt) one block at a time. In contrast, a stream cipher first uses the key to construct a ***keystream***, which is a bitstring that has exactly the same length as the plaintext (the plaintext is a bitstring of arbitrary length). The encryption operation constructs the ciphertext as the exclusive-or of the plaintext and the keystream. Decryption is accomplished by computing the exclusive-or of the ciphertext and the keystream. Public-key cryptosystems are invariably block ciphers, while secret-key cryptosystems can be block ciphers or stream ciphers. Block ciphers are studied in detail in Chapter 4.

### 1.1.4 Hybrid Cryptography

One of the drawbacks of public-key cryptosystems is that they are much slower than secret-key cryptosystems. As a consequence, public-key cryptosystems are mainly used to encrypt small amounts of data, e.g., a credit card number. However, there is a nice way to combine secret- and public-key cryptography to achieve the benefits of both. This technique is called ***hybrid cryptography***. Suppose that Alice wants to encrypt a "long" message and send it to Bob. Assume that Alice and Bob do not have a prior shared secret key. Alice can choose a random secret key and encrypt the plaintext, using a (fast) secret-key cryptosystem. Alice then encrypts this secret key using Bob's public key. Alice sends the ciphertext and the encrypted key to Bob. Bob first uses his private decryption key to decrypt the secret key, and then he uses this secret key to decrypt the ciphertext.

Notice that the "slow" public-key cryptosystem is only used to encrypt a short secret key. The much faster secret-key cryptosystem is used to encrypt the longer plaintext. Thus, hybrid cryptography (almost) achieves the efficiency of secret-key cryptography, but it can be used in a situation where Alice and Bob do not have a previously determined secret key.

## 1.2 Message Integrity

This section discusses various tools that help to achieve integrity of data, including message authentication codes (MACs), signature schemes, and hash functions.

Cryptosystems provide ***secrecy*** (equivalently, ***confidentiality***) against an eavesdropping adversary, which is often called a ***passive adversary***. A passive adversary is assumed to be able to access whatever information is being sent from Alice to Bob; see Figure 1.1. However, there are many other threats that we might want to protect against, particularly when an ***active adversary*** is present. An active adversary is one who can alter information that is transmitted from Alice to Bob.

Figure 1.2 depicts some of the possible actions of an active adversary. An active adversary might

- alter the information that is sent from Alice to Bob,
- send information to Bob in such a way that Bob thinks the information originated from Alice, or
- divert information sent from Alice to Bob in such a way that a third party (Charlie) receives this information instead of Bob.

Possible objectives of an active adversary could include fooling Bob (say) into accepting "bogus" information, or misleading Bob as to who sent the information to him in the first place.

We should note that encryption, by itself, cannot protect against these kinds of active attacks. For example, a stream cipher is susceptible to a ***bit-flipping attack***. If some ciphertext bits are "flipped" (i.e., 0's are replaced by 1's and vice versa), then the effect is to flip the corresponding plaintext bits. Thus, an adversary can modify the plaintext in a predictable way, even though the adversary does not know what the plaintext bits are.

There are various types of "integrity" guarantees that we might seek to provide, in order to protect against the possible actions of an active adversary. Such an adversary might change the information that is being transmitted from Alice to Bob (and note that this information may or may not be encrypted). Alternatively, the adversary might try to "forge" a message and send it to Bob, hoping that he will think that it originated from Alice. Cryptographic tools that protect against these and related types of threats can be constructed in both the secret-key and public-key settings. In the secret-key setting, we will briefly discuss the notion of a ***message authentication code*** (or ***MAC***). In the public-key setting, the tool that serves a roughly similar purpose is a ***signature scheme***.

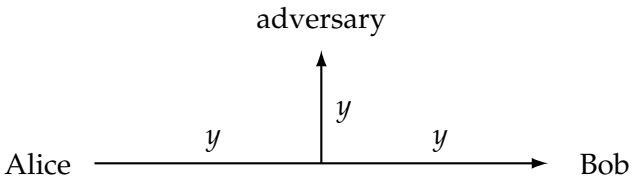


**FIGURE 1.1**: A passive adversary

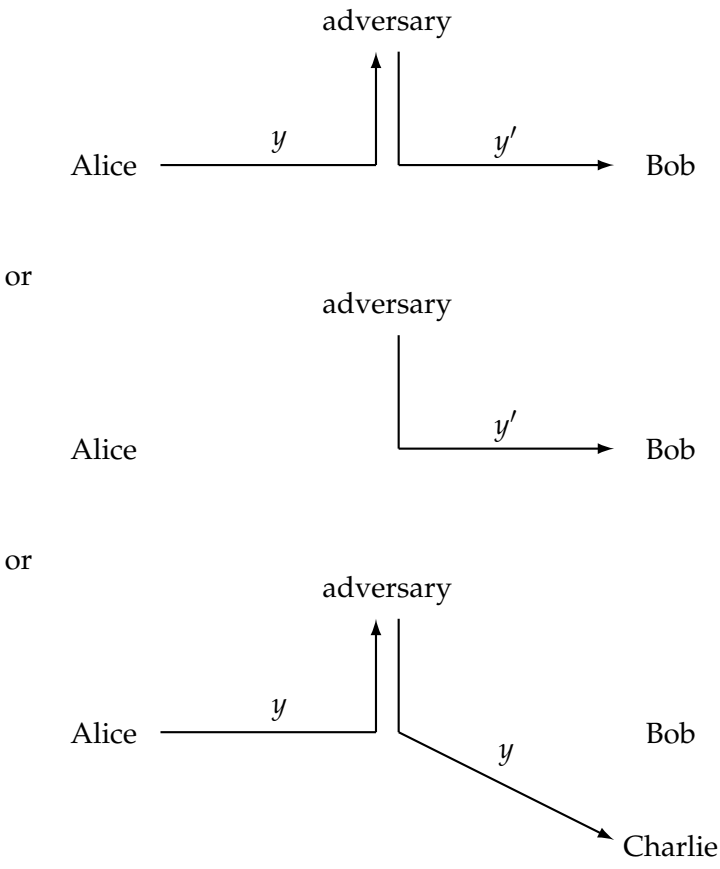


**FIGURE 1.2**: Active adversaries

### 1.2.1 Message Authentication Codes

A message authentication code requires Alice and Bob to share a secret key. When Alice wants to send a message to Bob, she uses the secret key to create a ***tag*** that she appends to the message (the tag depends on both the key and the message). When Bob receives the message and tag, he uses the key to re-compute the tag and checks to see if it is the same as the tag that he received. If so, Bob accepts the message as an authentic message from Alice; if not, then Bob rejects the message as being invalid. We note that the message may or may not be encrypted. MACs are discussed in Chapter 5.

If there is no need for confidentiality, then the message can be sent as plaintext. However, if confidentiality is desired, then the plaintext would be encrypted, and then the tag would be computed on the ciphertext. Bob would first verify the correctness of the tag. If the tag is correct, Bob would then decrypt the ciphertext. This process is often called ***encrypt-then-MAC*** (see Section 5.5.3 for a more detailed discussion of this topic).

For a MAC to be considered secure, it should be infeasible for the adversary to compute a correct tag for any message for which they have not already seen a valid tag. Suppose we assume that a secure MAC is being employed by Alice and Bob (and suppose that the adversary does not know the secret key that they are using). Then, if Bob receives a message and a valid tag, he can be confident that Alice created the tag on the given message (provided that Bob did not create it himself) and that neither the message nor the tag was altered by an adversary. A similar conclusion can be reached by Bob when he receives a message from Alice, along with a correct tag.

### 1.2.2 Signature Schemes

In the public-key setting, a signature scheme provides assurance similar to that provided by a MAC. In a signature scheme, the private key specifies a ***signing algorithm*** that Alice can use to sign messages. Similar to a MAC, the signing algorithm produces an output, which in this case is called a ***signature***, that depends on the message being signed as well as the key. The signature is then appended to the message. Notice that the signing algorithm is known only to Alice. On the other hand, there is a ***verification algorithm*** that is a public key (known to everyone). The verification algorithm takes as input a message and a signature, and outputs *true* or *false* to indicate whether the signature should be accepted as valid. One nice feature of a signature scheme is that anyone can verify Alice's signatures on messages, provided that they have an authentic copy of Alice's verification key. In contrast, in the MAC setting, only Bob can verify tags created by Alice (when Alice and Bob share a secret key). Signature schemes are studied in Chapter 8.

Security requirements for signature schemes are similar to MACs. It should be infeasible for an adversary to create a valid signature on any message not previously signed by Alice. Therefore, if Bob (or anyone else) receives a message and a valid tag (i.e., one that can be verified using Alice's public verification algorithm),

then the recipient can be confident that the signature was created by Alice and neither the message nor the signature was modified by an adversary.

One common application of signatures is to facilitate secure software updates. When a user purchases software from an online website, it typically includes a verification algorithm for a signature scheme. Later, when an updated version of the software is downloaded, it includes a signature (on the updated software). This signature can be verified using the verification algorithm that was downloaded when the original version of the software was purchased. This enables the user's computer to verify that the update comes from the same source as the original version of the software.

Signature schemes can be combined with public-key encryption schemes to provide confidentiality along with the integrity guarantees of a signature scheme. Assume that Alice wants to send a signed, encrypted (short) message to Bob. In this situation, the most commonly used technique is for Alice to first create a signature on the plaintext using her private signing algorithm, and then encrypt the plaintext and signature using Bob's public encryption key. When Bob receives the message, he first decrypts it, and then he checks the validity of the signature. This process is called ***sign-then-encrypt***; note that this is in some sense the reverse of the "encrypt-then-MAC" procedure that is used in the secret-key setting.

### 1.2.3 Nonrepudiation

There is one somewhat subtle difference between MACs and signature schemes. In a signature scheme, the verification algorithm is public. This means that the signature can be verified by anyone. So, if Bob receives a message from Alice containing her valid signature on the message, he can show the message and the signature to anyone else and be confident that the third party will also accept the signature as being valid. Consequently, Alice cannot sign a message and later try to claim that she did not sign the message, a property that is termed ***nonrepudiation***. This is useful in the setting of contracts, where we do not want someone to be able to renege on a signed contract by claiming (falsely) that their signature has been "forged," for example.

However, for a MAC, there is no third-party verifiability because the secret key is required to verify the correctness of the tag, and the key is known only to Alice and Bob. Even if the secret key is revealed to a third party (e.g., as a result of a court order), there is no way to determine if the tag was created by Alice or by Bob, because anything Bob can do, Alice can do as well, and vice versa. So a MAC does not provide nonrepudiation, and for this reason, a MAC is sometimes termed "deniable." It is interesting to note, however, that there are situations where deniability is desirable. This could be the case in real-time communications, where Alice and Bob want to be assured of the authenticity of their communications as they take place, but they do not want a permanent, verifiable record of this communication to exist. Such communication is analogous to an "off-the-record" conversation, e.g., between a journalist and an anonymous source. A MAC is useful in the con-

text of conversations of this type, especially if care is taken, after the conversation is over, to delete the secret keys that are used during the communication.

### 1.2.4 Certificates

We mentioned that verifying the authenticity of public keys, before they are used, is important. A certificate is a common tool to help achieve this objective. A ***certificate*** will contain information about a particular user or, more commonly, a website, including the website's public keys. These public keys will be signed by a trusted authority. It is assumed that everyone has possession of the trusted authority's public verification key, so anyone can verify the trusted authority's signature on a certificate. See Section 8.6 for more information about certificates.

This technique is used on the internet in *Transport Layer Security* (which is commonly called *TLS*). When a user connects to a secure website, say one belonging to a business engaged in electronic commerce, the website of the company will send a certificate to the user so the user can verify the authenticity of the website's public keys. These public keys will subsequently be used to set up a secure channel, between the user and the website, in which all information is encrypted. Note that the public key of the trusted authority, which is used to verify the public key of the website, is typically hard-coded into the web browser.

### 1.2.5 Hash Functions

Signature schemes tend to be much less efficient than MACs. So it is not advisable to use a signature scheme to sign "long" messages. (Actually, most signature schemes are designed to only sign messages of a short, fixed length.) In practice, messages are "hashed" before they are signed. A ***cryptographic hash function*** is used to compress a message of arbitrary length to a short, random-looking, fixed-length ***message digest***. Note that a hash function is a public function that is assumed to be known to everyone. Further, a hash function has no key. Hash functions are discussed in Chapter 5.

After Alice hashes the message, she signs the message digest, using her private signing algorithm. The original message, along with the signature on the message, is then transmitted to Bob, say. This process is called ***hash-then-sign***. To verify the signature, Bob will compute the message digest by hashing the message. Then he will use the public verification algorithm to check the validity of the signature on the message digest. When a signature is used along with public-key encryption, the process would actually be ***hash-then-sign-then-encrypt***. That is, the message is hashed, the message digest is then signed, and finally, the message and signature are encrypted.

A cryptographic hash function is very different from a hash function that is used to construct a hash table, for instance. In the context of hash tables, a hash function is generally required only to yield collisions[1] with a sufficiently small probability. On the other hand, if a cryptographic hash function is used, it should

[1] A ***collision*** for a function $h$ occurs when $h(x) = h(y)$ for some $x \neq y$.

be computationally infeasible to find collisions, even though they must exist. Cryptographic hash functions are usually required to satisfy additional security properties, as discussed in Section 5.2.

Cryptographic hash functions also have other uses, such as for ***key derivation***. When used for key derivation, a hash function would be applied to a long random string in order to create a short random key.

Finally, it should be emphasized that hash functions cannot be used for encryption, for two fundamental reasons. First is the fact that hash functions do not have a key. The second is that hash functions cannot be inverted (they are not injective functions) so a message digest cannot be "decrypted" to yield a unique plaintext value.

---

## 1.3 Cryptographic Protocols

Cryptographic tools such as cryptosystems, signature schemes, hash functions, etc., can be used on their own to achieve specific security objectives. However, these tools are also used as components in more complicated protocols. (Of course, protocols can also be designed "from scratch," without making use of prior primitives.)

In general, a ***protocol*** (or ***interactive protocol***) refers to a specified sequence of messages exchanged between two (or possibly more) parties. A ***session*** of a protocol between Alice and Bob, say, will consist of one or more ***flows***, where each flow consists of a message sent from Alice to Bob or vice versa. At the end of the session, the parties involved may have established some common shared information, or confirmed possession of some previously shared information.

One important type protocol is an ***identification scheme***, in which one party "proves" their identity to another by demonstrating possession of a password, for example. More sophisticated identification protocols will instead consist of two (or more) flows, for example a challenge followed by a response, where the response is computed from the challenge using a certain secret or private key. Identification schemes are the topic of Chapter 10.

There are many kinds of protocols associated with various aspects of choosing keys or communicating keys from one party to another. In a ***key distribution scheme***, keys might be chosen by a trusted authority and communicated to one or more members of a certain network. Another approach, which does not require the participation of an active trusted authority, is called ***key agreement***. In a key agreement scheme, Alice and Bob (say) are able to end up with a common shared secret key, which should not become known to an adversary. These and related topics are discussed in Chapters 11 and 12.

A ***secret sharing scheme*** involves a trusted authority distributing "pieces" of information (called "shares") in such a way that certain subsets of shares can be suitably combined to reconstruct a certain predefined secret. One common type

of secret sharing scheme is a ***threshold scheme***. In a $(k, n)$-threshold scheme, there are $n$ shares, and any $k$ shares permit the reconstruction of the secret. On the other hand, $k-1$ or fewer shares provide no information about the value of the secret. Secret sharing schemes are studied in Chapter 11.

---

## 1.4 Security

A fundamental goal for a cryptosystem, signature scheme, etc., is for it to be "secure." But what does it mean to be secure and how can we gain confidence that something is indeed secure? Roughly speaking, we would want to say that an adversary cannot succeed in "breaking" a cryptosystem, for example, but we have to make this notion precise. Security in cryptography involves consideration of three different aspects: an ***attack model***, an ***adversarial goal***, and a ***security level***. We will discuss each of these in turn.

The attack model specifies the information that is available to the adversary. We will always assume that the adversary knows the scheme or protocol being used (this is called ***Kerckhoffs' Principle***). The adversary is also assumed to know the public key (if the system is a public-key system). On the other hand, the adversary is assumed not to know any secret or private keys being used. Possible additional information provided to the adversary should be specified in the attack model.

The adversarial goal specifies exactly what it means to "break" the cryptosystem. What is the adversary attempting to do and what information are they trying to determine? Thus, the adversarial goal defines a "successful attack."

The security level attempts to quantify the effort required to break the cryptosystem. Equivalently, what computational resources does the adversary have access to and how much time would it take to carry out an attack using those resources?

A statement of security for a cryptographic scheme will assert that a particular adversarial goal cannot be achieved in a specified attack model, given specified computational resources.

We now illustrate some of the above concepts in relation to a cryptosystem. There are four commonly considered attack models. In a ***known ciphertext attack***, the adversary has access to some amount of ciphertext that is all encrypted with the same unknown key. In a ***known plaintext attack***, the adversary gains access to some plaintext as well as the corresponding ciphertext (all of which is encrypted with the same key). In a ***chosen plaintext attack***, the adversary is allowed to choose plaintext, and then they are given the corresponding ciphertext. Finally, in a ***chosen ciphertext attack***, the adversary chooses some ciphertext and they are then given the corresponding plaintext.

Clearly a chosen plaintext or chosen ciphertext attack provides the adversary with more information than a known ciphertext attack. So they would be con-

sidered to be stronger attack models than a known ciphertext attack, since they potentially make the adversary's job easier.

The next aspect to study is the adversarial goal. In a ***complete break*** of a cryptosystem, the adversary determines the private (or secret) key. However, there are other, weaker goals that the adversary could potentially achieve, even if a complete break is not possible. For example, the adversary might be able to decrypt a previously unseen ciphertext with some specified non-zero probability, even though they have not been able to determine the key. Or, the adversary might be able to determine some partial information about the plaintext, given a previously unseen ciphertext, with some specified non-zero probability. "Partial information" could include the values of certain plaintext bits. Finally, as an example of a weak goal, the adversary might be able distinguish between encryptions of two given plaintexts.[2]

Other cryptographic primitives will have different attack models and adversarial goals. In a signature scheme, the attack model would specify what kind of (valid) signatures the adversary has access to. Perhaps the adversary just sees some previously signed messages, or maybe the adversary can request the signer to sign some specific messages of the adversary's choosing. The adversarial goal is typically to sign some "new" message (i.e., one for which the adversary does not already know a valid signature). Perhaps the adversary can find a valid signature for some specific message that the adversary chooses, or perhaps they can find a valid signature for any message. These would represent weak and strong adversarial goals, respectively.

Three levels of security are often studied, which are known as ***computational security***, ***provable security***, and ***unconditional security***.

Computational security means that a specific algorithm to break the system is computationally infeasible, i.e., it cannot be accomplished in a reasonable amount of time using currently available computational resources. Of course, a system that is computationally secure today may not be computationally secure indefinitely. For example, new algorithms might be discovered, computers may get faster, or fundamental new computing paradigms such as quantum computing might become practical. Quantum computing, if it becomes practical, could have an enormous impact on the security of many kinds of public-key cryptography; this is addressed in more detail in Section 9.1.

It is in fact very difficult to predict how long something that is considered secure today will remain secure. There are many examples where many cryptographic schemes have not survived as long as originally expected due to the reasons mentioned above. This has led to rather frequent occurrences of replacing standards with improved standards. For example, in the case of hash functions, there have been a succession of proposed and/or approved standards, denoted as *SHA-0*, *SHA-1*, *SHA-2* and *SHA-3*, as new attacks have been found and old standards have become insecure.

[2] Whether or not this kind of limited information can be exploited by the adversary in a malicious way is another question, of course.

An interesting example relating to broken predictions is provided by the public-key *RSA Cryptosystem*. In the August 1977 issue of *Scientific American*, the eminent mathematical expositor Martin Gardner wrote a column on the newly developed *RSA* public-key cryptosystem entitled "A new kind of cipher that would take millions of years to break." Included in the article was a challenge ciphertext, encrypted using a 512-bit key. However, the challenge was solved 17 years later, on April 26, 1994, by factoring the given public key (the plaintext was "the magic words are squeamish ossifrage"). The statement that the cipher would take millions of years to break probably referred to how long it would take to run the best factoring algorithm known in 1977 on the fastest computer available in 1977. However, between 1977 and 1994, there were several developments, including the following:

- computers became much faster,
- improved factoring algorithms were found, and
- the development of the internet facilitated large-scale distributed computations.

Of course, it is basically impossible to predict when new algorithms will be discovered. Also, the third item listed above can be regarded as a "paradigm shift" that was probably not on anyone's radar in 1977.

The next "level" of security we address is provable security (also known as ***reductionist security***), which refers to a situation where breaking the cryptosystem (i.e., achieving the adversarial goal) can be reduced in a complexity-theoretic sense to solving some underlying (assumed difficult) mathematical problem. This would show that breaking the cryptosystem is at least as difficult as solving the given hard problem. Provable security often involves reductions to the factoring problem or the discrete logarithm problem (these problems are studied in Sections 6.6 and 7.2, respectively).

Finally, unconditional security means that the cryptosystem cannot be broken (i.e., the adversarial goal is not achievable), even with unlimited computational resources, because there is not enough information available to the adversary (as specified in the attack model) for them to be able to do this. The most famous example of an unconditionally secure cryptosystem is the *One-time Pad*. In this cryptosystem, the key is a random bitstring having the same length as the plaintext. The ciphertext is formed as the exclusive-or of the plaintext and the key. For the *One-time Pad*, it can be proven mathematically that the adversary can obtain no partial information whatsoever about the plaintext (other than its length), given the ciphertext, provided the key is used to encrypt only one string of plaintext and the key has the same length as the plaintext. The *One-time Pad* is discussed in Chapter 3.

When we analyze a cryptographic scheme, our goal would be to show that the adversary cannot achieve a *weak* adversarial goal in a *strong* attack model, given *significant* computational resources.

The preceding discussion of security has dealt mostly with the situation of a cryptographic primitive such as a cryptosystem. However, cryptographic primitives are generally combined in complicated ways when protocols are defined and ultimately implemented. Even seemingly simple implementation decisions can lead to unexpected vulnerabilities. For example, when data is encrypted using a block cipher, it first needs to be split into fixed length chunks, e.g., 128-bit blocks. If the data does not exactly fill up an integral number of blocks, then some padding has to be introduced. It turns out that a standard padding technique, when used with the common CBC mode of operation, is susceptible to an attack known as a ***padding oracle attack***, which was discovered by Vaudenay in 2002 (see Section 4.7.1 for a description of this attack).

There are also various kinds of attacks against physical implementations of cryptography that are known as ***side channel attacks***. Examples of these include ***timing attacks***, ***fault analysis attacks***, ***power analysis attacks***, and ***cache attacks***. The idea is that information about a secret or private key might be leaked by observing or physically manipulating a device (such as a smart card) on which a particular cryptographic scheme is implemented. One example would be observing the time taken by the device to perform certain computations (a so-called "timing attack"). This leakage of information can take place even though the scheme is "secure."

---

## 1.5 Notes and References

There are many monographs and textbooks on the subject of cryptography. We will mention here a few general treatments that may be useful to readers.

For an accessible, non-mathematical treatment, we recommend

- *Everyday Cryptography: Fundamental Principles and Applications, Second Edition* by Keith Martin [127].

For a more mathematical point of view, the following recent texts are helpful:

- *An Introduction to Mathematical Cryptography* by J. Hoffstein, J. Pipher, and J. Silverman [96]
- *Introduction to Modern Cryptography, Second Edition* by J. Katz and Y. Lindell [104]
- *Understanding Cryptography: A Textbook for Students and Practitioners* by C. Paar and J. Pelzl [157]
- *Cryptography Made Simple* by Nigel Smart [185]
- *A Classical Introduction to Cryptography: Applications for Communications Security* by Serge Vaudenay [196].

For mathematical background, especially for public-key cryptography, we recommend

- *Mathematics of Public Key Cryptography* by Stephen Galbraith [84].

Finally, the following is a valuable reference, even though it is quite out of date:

- *Handbook of Applied Cryptography* by A.J. Menezes, P.C. Van Oorschot, and S.A. Vanstone [134].

# Chapter 2

# Classical Cryptography

In this chapter, we provide a gentle introduction to cryptography and cryptanalysis. We present several simple systems, and describe how they can be "broken." Along the way, we discuss various mathematical techniques that will be used throughout the book.

## 2.1 Introduction: Some Simple Cryptosystems

The fundamental objective of cryptography is to enable two people, usually referred to as ***Alice*** and ***Bob***, to communicate over an insecure ***channel*** in such a way that an opponent, ***Oscar***, cannot understand what is being said. This channel could be a telephone line or computer network, for example. The information that Alice wants to send to Bob, which we call "plaintext," can be English text, numerical data, or anything at all—its structure is completely arbitrary. Alice encrypts the plaintext, using a predetermined key, and sends the resulting ciphertext over the channel. Oscar, upon seeing the ciphertext in the channel by eavesdropping, cannot determine what the plaintext was; but Bob, who knows the encryption key, can decrypt the ciphertext and reconstruct the plaintext.

These ideas are described formally using the following mathematical notation.

**Definition 2.1:** A ***cryptosystem*** is a five-tuple $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$, where the following conditions are satisfied:

1. $\mathcal{P}$ is a finite set of possible ***plaintexts***;
2. $\mathcal{C}$ is a finite set of possible ***ciphertexts***;
3. $\mathcal{K}$, the ***keyspace***, is a finite set of possible ***keys***;
4. For each $K \in \mathcal{K}$, there is an ***encryption rule*** $e_K \in \mathcal{E}$ and a corresponding ***decryption rule*** $d_K \in \mathcal{D}$. Each $e_K : \mathcal{P} \to \mathcal{C}$ and $d_K : \mathcal{C} \to \mathcal{P}$ are functions such that $d_K(e_K(x)) = x$ for every plaintext element $x \in \mathcal{P}$.

The main property is property 4. It says that if a plaintext $x$ is encrypted using $e_K$, and the resulting ciphertext is subsequently decrypted using $d_K$, then the original plaintext $x$ results.

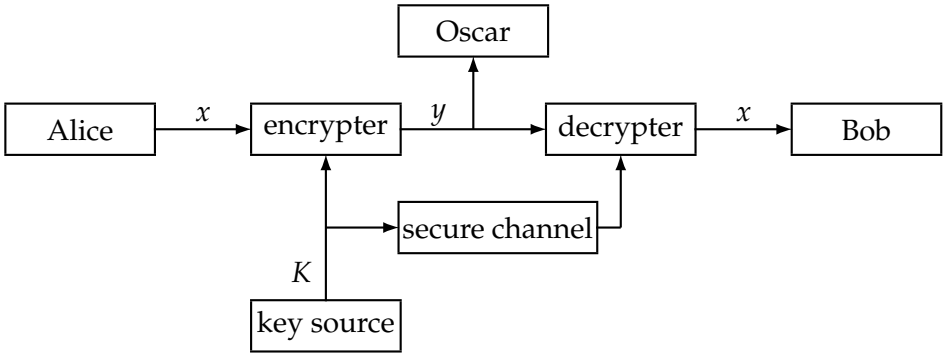


**FIGURE 2.1**: The communication channel

Alice and Bob will employ the following protocol to use a specific cryptosystem. First, they choose a random key $K \in \mathcal{K}$. This is done when they are in the same place and are not being observed by Oscar, or, alternatively, when they do have access to a secure channel, in which case they can be in different places. At a later time, suppose Alice wants to communicate a message to Bob over an insecure channel. We suppose that this message is a ***string***

$$\mathbf{x} = x_1 x_2 \cdots x_n$$

for some integer $n \geq 1$, where each plaintext symbol $x_i \in \mathcal{P}$, $1 \leq i \leq n$. Each $x_i$ is encrypted using the encryption rule $e_K$ specified by the predetermined key $K$. Hence, Alice computes $y_i = e_K(x_i)$, $1 \leq i \leq n$, and the resulting ciphertext string

$$\mathbf{y} = y_1 y_2 \cdots y_n$$

is sent over the channel. When Bob receives $y_1 y_2 \cdots y_n$, he decrypts it using the decryption function $d_K$, obtaining the original plaintext string, $x_1 x_2 \cdots x_n$. See Figure 2.1 for an illustration of the communication channel.

Clearly, it must be the case that each encryption function $e_K$ is an ***injective function*** (i.e., one-to-one); otherwise, decryption could not be accomplished in an unambiguous manner. For example, if

$$y = e_K(x_1) = e_K(x_2)$$

where $x_1 \neq x_2$, then Bob has no way of knowing whether $y$ should decrypt to $x_1$ or $x_2$. Note that if $\mathcal{P} = \mathcal{C}$, it follows that each encryption function is a permutation. That is, if the set of plaintexts and ciphertexts are identical, then each encryption function just rearranges (or permutes) the elements of this set.

### 2.1.1 The Shift Cipher

In this section, we will describe the *Shift Cipher*, which is based on modular arithmetic. But first we review some basic definitions of ***modular arithmetic***.

**Definition 2.2:** Suppose $a$ and $b$ are integers, and $m$ is a positive integer. Then we write $a \equiv b \pmod{m}$ if $m$ divides $b - a$. The phrase $a \equiv b \pmod{m}$ is called a ***congruence***, and it is read as "$a$ is ***congruent*** to $b$ modulo $m$." The integer $m$ is called the ***modulus***.

Suppose we divide $a$ and $b$ by $m$, obtaining integer quotients and remainders, where the remainders are between 0 and $m-1$. That is, $a = q_1 m + r_1$ and $b = q_2 m + r_2$, where $0 \leq r_1 \leq m-1$ and $0 \leq r_2 \leq m-1$. Then it is not difficult to see that $a \equiv b \pmod{m}$ if and only if $r_1 = r_2$. We will use the notation $a \bmod m$ (without parentheses) to denote the remainder when $a$ is divided by $m$, i.e., the value $r_1$ above. Thus $a \equiv b \pmod{m}$ if and only if $a \bmod m = b \bmod m$. If we replace $a$ by $a \bmod m$, we say that $a$ is ***reduced modulo*** $m$.

We give a couple of examples. To compute $101 \bmod 7$, we write $101 = 7 \times 14 + 3$. Since $0 \leq 3 \leq 6$, it follows that $101 \bmod 7 = 3$. As another example, suppose we want to compute $(-101) \bmod 7$. In this case, we write $-101 = 7 \times (-15) + 4$. Since $0 \leq 4 \leq 6$, it follows that $(-101) \bmod 7 = 4$.

**REMARK** Many computer programming languages define $a \bmod m$ to be the remainder in the range $-m+1, \ldots, m-1$ having the same sign as $a$. For example, $(-101) \bmod 7$ would be $-3$, rather than 4 as we defined it above. But for our purposes, it is much more convenient to define $a \bmod m$ always to be non-negative. ∎

We now define arithmetic modulo $m$: $\mathbb{Z}_m$ is the set $\{0, \ldots, m-1\}$, equipped with two operations, $+$ and $\times$. Addition and multiplication in $\mathbb{Z}_m$ work exactly like real addition and multiplication, except that the results are reduced modulo $m$.

For example, suppose we want to compute $11 \times 13$ in $\mathbb{Z}_{16}$. As integers, we have $11 \times 13 = 143$. Then we reduce 143 modulo 16 as described above: $143 = 8 \times 16 + 15$, so $143 \bmod 16 = 15$, and hence $11 \times 13 = 15$ in $\mathbb{Z}_{16}$.

These definitions of addition and multiplication in $\mathbb{Z}_m$ satisfy most of the familiar rules of arithmetic. We will list these properties now, without proof:

1. addition is ***closed***, i.e., for any $a, b \in \mathbb{Z}_m$, $a + b \in \mathbb{Z}_m$
2. addition is ***commutative***, i.e., for any $a, b \in \mathbb{Z}_m$, $a + b = b + a$
3. addition is ***associative***, i.e., for any $a, b, c \in \mathbb{Z}_m$, $(a + b) + c = a + (b + c)$
4. 0 is an ***additive identity***, i.e., for any $a \in \mathbb{Z}_m$, $a + 0 = 0 + a = a$
5. the ***additive inverse*** of any $a \in \mathbb{Z}_m$ is $m - a$, i.e., $a + (m - a) = (m - a) + a = 0$ for any $a \in \mathbb{Z}_m$

**Cryptosystem 2.1:** *Shift Cipher*

Let $\mathcal{P} = \mathcal{C} = \mathcal{K} = \mathbb{Z}_{26}$. For $0 \leq K \leq 25$, define

$$e_K(x) = (x + K) \bmod 26$$

and

$$d_K(y) = (y - K) \bmod 26$$

$(x, y \in \mathbb{Z}_{26})$.

6. multiplication is ***closed***, i.e., for any $a, b \in \mathbb{Z}_m$, $ab \in \mathbb{Z}_m$
7. multiplication is ***commutative***, i.e., for any $a, b \in \mathbb{Z}_m$, $ab = ba$
8. multiplication is ***associative***, i.e., for any $a, b, c \in \mathbb{Z}_m$, $(ab)c = a(bc)$
9. 1 is a ***multiplicative identity***, i.e., for any $a \in \mathbb{Z}_m$, $a \times 1 = 1 \times a = a$
10. the ***distributive property*** is satisfied, i.e., for any $a, b, c \in \mathbb{Z}_m$, $(a + b)c = (ac) + (bc)$ and $a(b + c) = (ab) + (ac)$.

Properties 1, 3–5 say that $\mathbb{Z}_m$ forms an algebraic structure called a ***group*** with respect to the addition operation. Since property 2 also holds, the group is said to be an ***abelian group***.

Properties 1–10 establish that $\mathbb{Z}_m$ is, in fact, a ***ring***. We will see many other examples of groups and rings in this book. Some familiar examples of rings include the integers, $\mathbb{Z}$; the real numbers, $\mathbb{R}$; and the complex numbers, $\mathbb{C}$. However, these are all infinite rings, and our attention will be confined almost exclusively to finite rings.

Since additive inverses exist in $\mathbb{Z}_m$, we can also subtract elements in $\mathbb{Z}_m$. We define $a - b$ in $\mathbb{Z}_m$ to be $(a - b) \bmod m$. That is, we compute the integer $a - b$ and then reduce it modulo $m$. For example, to compute $11 - 18$ in $\mathbb{Z}_{31}$, we first subtract 18 from 11, obtaining $-7$, and then compute $(-7) \bmod 31 = 24$.

We present the *Shift Cipher* as Cryptosystem 2.1. It is defined over $\mathbb{Z}_{26}$ since there are 26 letters in the English alphabet, though it could be defined over $\mathbb{Z}_m$ for any modulus $m$. It is easy to see that the *Shift Cipher* forms a cryptosystem as defined above, i.e., $d_K(e_K(x)) = x$ for every $x \in \mathbb{Z}_{26}$.

**REMARK** For the particular key $K = 3$, the cryptosystem is often called the *Caesar Cipher*, which was purportedly used by Julius Caesar. ∎

We would use the *Shift Cipher* (with a modulus of 26) to encrypt ordinary English text by setting up a correspondence between alphabetic characters and

residues modulo 26 as follows: $A \leftrightarrow 0$, $B \leftrightarrow 1, \ldots, Z \leftrightarrow 25$. Since we will be using this correspondence in several examples, let's record it for future use:

| $A$ | $B$ | $C$ | $D$ | $E$ | $F$ | $G$ | $H$ | $I$ | $J$ | $K$ | $L$ | $M$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |

| $N$ | $O$ | $P$ | $Q$ | $R$ | $S$ | $T$ | $U$ | $V$ | $W$ | $X$ | $Y$ | $Z$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 |

A small example will illustrate.

***Example 2.1*** Suppose the key for a *Shift Cipher* is $K = 11$, and the plaintext is

`wewillmeetatmidnight`.

We first convert the plaintext to a sequence of integers using the specified correspondence, obtaining the following:

| 22 | 4 | 22 | 8 | 11 | 11 | 12 | 4 | 4 | 19 |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 19 | 12 | 8 | 3 | 13 | 8 | 6 | 7 | 19 |

Next, we add 11 to each value, reducing each sum modulo 26:

| 7 | 15 | 7 | 19 | 22 | 22 | 23 | 15 | 15 | 4 |
|---|---|---|---|---|---|---|---|---|---|
| 11 | 4 | 23 | 19 | 14 | 24 | 19 | 17 | 18 | 4 |

Finally, we convert the sequence of integers to alphabetic characters, obtaining the ciphertext:

`HPHTWWXPPELEXTOYTRSE`.

To decrypt the ciphertext, Bob will first convert the ciphertext to a sequence of integers, then subtract 11 from each value (reducing modulo 26), and finally convert the sequence of integers to alphabetic characters. □

**REMARK** In the above example we are using upper case letters for ciphertext and lower case letters for plaintext, in order to improve readability. We will do this elsewhere as well. ▮

If a cryptosystem is to be of practical use, it should satisfy certain properties. We informally enumerate two of these properties now.

1. Each encryption function $e_K$ and each decryption function $d_K$ should be efficiently computable.

2. An opponent, upon seeing a ciphertext string $\mathbf{y}$, should be unable to determine the key $K$ that was used, or the plaintext string $\mathbf{x}$.

The second property is defining, in a very vague way, the idea of "security." The process of attempting to compute the key $K$, given a string of ciphertext $\mathbf{y}$, is called ***cryptanalysis***. (We will make these concepts more precise as we proceed.) Note that, if Oscar can determine $K$, then he can decrypt $\mathbf{y}$ just as Bob would, using $d_K$. Hence, determining $K$ is at least as difficult as determining the plaintext string $\mathbf{x}$, given the ciphertext string $\mathbf{y}$.

We observe that the *Shift Cipher* (modulo 26) is not secure, since it can be cryptanalyzed by the obvious method of ***exhaustive key search***. Since there are only 26 possible keys, it is easy to try every possible decryption rule $d_K$ until a "meaningful" plaintext string is obtained. This is illustrated in the following example.

***Example 2.2*** Given the ciphertext string

```
JBCRCLQRWCRVNBJENBWRWN,
```

we successively try the decryption keys $d_0, d_1$, etc. The following is obtained:

```
jbcrclqrwcrvnbjenbwrwn
iabqbkpqvbqumaidmavqvm
hzapajopuaptlzhclzupul
gyzozinotzoskygbkytotk
fxynyhmnsynrjxfajxsnsj
ewxmxglmrxmqiweziwrmri
dvwlwfklqwlphvdyhvqlqh
cuvkvejkpvkogucxgupkpg
btujudijoujnftbwftojof
astitchintimesavesnine
```

At this point, we have determined the plaintext to be the phrase "a stitch in time saves nine," and we can stop. The key is $K = 9$. □

On average, a plaintext will be computed using this method after trying $26/2 = 13$ decryption rules.

As the above example indicates, a necessary condition for a cryptosystem to be secure is that an exhaustive key search should be infeasible; i.e., the keyspace should be very large. As might be expected, however, a large keyspace is not sufficient to guarantee security.

### 2.1.2 The Substitution Cipher

Another well-known cryptosystem is the *Substitution Cipher*, which we define now. This cryptosystem has been used for hundreds of years. Puzzle "cryptograms" in newspapers are examples of *Substitution Ciphers*. This cipher is defined as Cryptosystem 2.2.

Actually, in the case of the *Substitution Cipher*, we might as well take $\mathcal{P}$ and $\mathcal{C}$ both to be the 26-letter English alphabet. We used $\mathbb{Z}_{26}$ in the *Shift Cipher* because

**Cryptosystem 2.2:** *Substitution Cipher*

Let $\mathcal{P} = \mathcal{C} = \mathbb{Z}_{26}$. $\mathcal{K}$ consists of all possible permutations of the 26 symbols $0, 1, \ldots, 25$. For each permutation $\pi \in \mathcal{K}$, define

$$e_\pi(x) = \pi(x),$$

and define

$$d_\pi(y) = \pi^{-1}(y),$$

where $\pi^{-1}$ is the inverse permutation to $\pi$.

encryption and decryption were algebraic operations. But in the *Substitution Cipher*, it is more convenient to think of encryption and decryption as permutations of alphabetic characters.

Here is an example of a "random" permutation, $\pi$, which could comprise an encryption function. (As before, plaintext characters are written in lower case and ciphertext characters are written in upper case.)

| $a$ | $b$ | $c$ | $d$ | $e$ | $f$ | $g$ | $h$ | $i$ | $j$ | $k$ | $l$ | $m$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $X$ | $N$ | $Y$ | $A$ | $H$ | $P$ | $O$ | $G$ | $Z$ | $Q$ | $W$ | $B$ | $T$ |

| $n$ | $o$ | $p$ | $q$ | $r$ | $s$ | $t$ | $u$ | $v$ | $w$ | $x$ | $y$ | $z$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $S$ | $F$ | $L$ | $R$ | $C$ | $V$ | $M$ | $U$ | $E$ | $K$ | $J$ | $D$ | $I$ |

Thus, $e_\pi(a) = X$, $e_\pi(b) = N$, etc. The decryption function is the inverse permutation. This is formed by writing the second lines first, and then sorting in alphabetical order. The following is obtained:

| $A$ | $B$ | $C$ | $D$ | $E$ | $F$ | $G$ | $H$ | $I$ | $J$ | $K$ | $L$ | $M$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $d$ | $l$ | $r$ | $y$ | $v$ | $o$ | $h$ | $e$ | $z$ | $x$ | $w$ | $p$ | $t$ |

| $N$ | $O$ | $P$ | $Q$ | $R$ | $S$ | $T$ | $U$ | $V$ | $W$ | $X$ | $Y$ | $Z$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $b$ | $g$ | $f$ | $j$ | $q$ | $n$ | $m$ | $u$ | $s$ | $k$ | $a$ | $c$ | $i$ |

Hence, $d_\pi(A) = d$, $d_\pi(B) = l$, etc.

As an exercise, the reader might decrypt the following ciphertext using this decryption function:

```
MGZVYZLGHCMHJMYXSSFMNHAHYCDLMHA.
```

A key for the *Substitution Cipher* just consists of a permutation of the 26 alphabetic characters. The number of possible permutations is 26!, which is more than $4.0 \times 10^{26}$, a very large number. Thus, an exhaustive key search is infeasible, even for a computer. However, we shall see later that a *Substitution Cipher* can easily be cryptanalyzed by other methods.

### 2.1.3 The Affine Cipher

The *Shift Cipher* is a special case of the *Substitution Cipher*, which includes only 26 of the 26! possible permutations of 26 elements. Another special case of the *Substitution Cipher* is the *Affine Cipher*, which we describe now. In the *Affine Cipher*, we restrict the encryption functions to functions of the form

$$e(x) = (ax + b) \bmod 26,$$

$a, b \in \mathbb{Z}_{26}$. Such a function is called an ***affine function***; hence the name *Affine Cipher*. (Observe that when $a = 1$, we have a *Shift Cipher*.)

In order that decryption is possible, it is necessary to ask when an affine function is injective. In other words, for any $y \in \mathbb{Z}_{26}$, we want the congruence

$$ax + b \equiv y \pmod{26}$$

to have a unique solution for $x$. This congruence is equivalent to

$$ax \equiv y - b \pmod{26}.$$

Now, as $y$ varies over $\mathbb{Z}_{26}$, so, too, does $y - b$ vary over $\mathbb{Z}_{26}$. Hence, it suffices to study the congruence $ax \equiv y \pmod{26}$ ($y \in \mathbb{Z}_{26}$).

We claim that this congruence has a unique solution for every $y$ if and only if $\gcd(a, 26) = 1$ (where the gcd function denotes the greatest common divisor of its arguments). First, suppose that $\gcd(a, 26) = d > 1$. Then the congruence $ax \equiv 0 \pmod{26}$ has (at least) two distinct solutions in $\mathbb{Z}_{26}$, namely $x = 0$ and $x = 26/d$. In this case $e(x) = (ax + b) \bmod 26$ is not an injective function and hence not a valid encryption function.

For example, since $\gcd(4, 26) = 2$, it follows that $4x + 7$ is not a valid encryption function: $x$ and $x + 13$ will encrypt to the same value, for any $x \in \mathbb{Z}_{26}$.

Let's next suppose that $\gcd(a, 26) = 1$. Suppose for some $x_1$ and $x_2$ that

$$ax_1 \equiv ax_2 \pmod{26}.$$

Then

$$a(x_1 - x_2) \equiv 0 \pmod{26},$$

and thus

$$26 \mid a(x_1 - x_2).$$

We now make use of a fundamental property of integer division: if $\gcd(a, b) = 1$ and $a \mid bc$, then $a \mid c$. Since $26 \mid a(x_1 - x_2)$ and $\gcd(a, 26) = 1$, we must therefore have that

$$26 \mid (x_1 - x_2),$$

i.e., $x_1 \equiv x_2 \pmod{26}$.

At this point we have shown that, if $\gcd(a, 26) = 1$, then a congruence of the form $ax \equiv y \pmod{26}$ has, at most, one solution in $\mathbb{Z}_{26}$. Hence, if we let $x$ vary over $\mathbb{Z}_{26}$, then $ax \bmod 26$ takes on 26 distinct values modulo 26. That is, it takes on

every value exactly once. It follows that, for any $y \in \mathbb{Z}_{26}$, the congruence $ax \equiv y \pmod{26}$ has a unique solution for $x$.

There is nothing special about the number 26 in this argument. The following result can be proved in an analogous fashion.

**THEOREM 2.1** *The congruence $ax \equiv b \pmod{m}$ has a unique solution $x \in \mathbb{Z}_m$ for every $b \in \mathbb{Z}_m$ if and only if $\gcd(a, m) = 1$.*

Since $26 = 2 \times 13$, the values of $a \in \mathbb{Z}_{26}$ such that $\gcd(a, 26) = 1$ are $a = 1$, 3, 5, 7, 9, 11, 15, 17, 19, 21, 23, and 25. The parameter $b$ can be any element in $\mathbb{Z}_{26}$. Hence the *Affine Cipher* has $12 \times 26 = 312$ possible keys. (Of course, this is much too small to be secure.)

Let's now consider the general setting where the modulus is $m$. We need another definition from number theory.

**Definition 2.3:** Suppose $a \geq 1$ and $m \geq 2$ are integers. If $\gcd(a, m) = 1$, then we say that $a$ and $m$ are ***relatively prime***. The number of integers in $\mathbb{Z}_m$ that are relatively prime to $m$ is often denoted by $\phi(m)$ (this function is called the ***Euler phi-function***).

A well-known result from number theory gives the value of $\phi(m)$ in terms of the prime power factorization of $m$. (An integer $p > 1$ is ***prime*** if it has no positive divisors other than 1 and $p$. Every integer $m > 1$ can be factored as a product of powers of primes in a unique way. For example, $60 = 2^2 \times 3 \times 5$ and $98 = 2 \times 7^2$.)

We record the formula for $\phi(m)$ in the following theorem.

**THEOREM 2.2** *Suppose*

$$m = \prod_{i=1}^{n} {p_i}^{e_i},$$

*where the $p_i$'s are distinct primes and $e_i > 0$, $1 \leq i \leq n$. Then*

$$\phi(m) = \prod_{i=1}^{n} ({p_i}^{e_i} - p_i^{e_i - 1}).$$

It follows that the number of keys in the *Affine Cipher* over $\mathbb{Z}_m$ is $m\phi(m)$, where $\phi(m)$ is given by the formula above. (The number of choices for $b$ is $m$, and the number of choices for $a$ is $\phi(m)$, where the encryption function is $e(x) = ax + b$.) For example, suppose $m = 60$. We have

$$60 = 2^2 \times 3^1 \times 5^1$$

and hence

$$\phi(60) = (4 - 2) \times (3 - 1) \times (5 - 1) = 2 \times 2 \times 4 = 16.$$

The number of keys in the *Affine Cipher* is $60 \times 16 = 960$.

Let's now consider the decryption operation in the *Affine Cipher* with modulus $m = 26$. Suppose that $\gcd(a, 26) = 1$. To decrypt, we need to solve the congruence $y \equiv ax + b \pmod{26}$ for $x$. The discussion above establishes that the congruence will have a unique solution in $\mathbb{Z}_{26}$, but it does not give us an efficient method of finding the solution. What we require is an efficient algorithm to do this. Fortunately, some further results on modular arithmetic will provide us with the efficient decryption algorithm we seek.

We require the idea of a multiplicative inverse.

**Definition 2.4:** Suppose $a \in \mathbb{Z}_m$. The ***multiplicative inverse*** of $a$ modulo $m$, denoted $a^{-1} \bmod m$, is an element $a' \in \mathbb{Z}_m$ such that $aa' \equiv a'a \equiv 1 \pmod{m}$. If $m$ is fixed, we sometimes write $a^{-1}$ for $a^{-1} \bmod m$.

By similar arguments to those used above, it can be shown that $a$ has a multiplicative inverse modulo $m$ if and only if $\gcd(a, m) = 1$; and if a multiplicative inverse exists, it is unique modulo $m$. Also, observe that if $b = a^{-1}$, then $a = b^{-1}$. If $p$ is prime, then every non-zero element of $\mathbb{Z}_p$ has a multiplicative inverse. A ring in which this is true is called a ***field***.

In Section 6.2.1, we will describe an efficient algorithm for computing multiplicative inverses in $\mathbb{Z}_m$ for any $m$. However, in $\mathbb{Z}_{26}$, trial and error suffices to find the multiplicative inverses of the elements relatively prime to 26:

$$\begin{aligned} 1^{-1} &= 1, \\ 3^{-1} &= 9, \\ 5^{-1} &= 21, \\ 7^{-1} &= 15, \\ 11^{-1} &= 19, \\ 17^{-1} &= 23, \text{and} \\ 25^{-1} &= 25. \end{aligned}$$

(All of these can be verified easily. For example, $7 \times 15 = 105 \equiv 1 \pmod{26}$, so $7^{-1} = 15$ and $15^{-1} = 7$.)

Consider our congruence $y \equiv ax + b \pmod{26}$. This is equivalent to

$$ax \equiv y - b \pmod{26}.$$

Since $\gcd(a, 26) = 1$, $a$ has a multiplicative inverse modulo 26. Multiplying both sides of the congruence by $a^{-1}$, we obtain

$$a^{-1}(ax) \equiv a^{-1}(y - b) \pmod{26}.$$

By associativity of multiplication modulo 26, we have that

$$a^{-1}(ax) \equiv (a^{-1}a)x \equiv 1x \equiv x \pmod{26}.$$

**Cryptosystem 2.3:** *Affine Cipher*

Let $\mathcal{P} = \mathcal{C} = \mathbb{Z}_{26}$ and let

$$\mathcal{K} = \{(a, b) \in \mathbb{Z}_{26} \times \mathbb{Z}_{26} : \gcd(a, 26) = 1\}.$$

For $K = (a, b) \in \mathcal{K}$, define

$$e_K(x) = (ax + b) \bmod 26$$

and

$$d_K(y) = a^{-1}(y - b) \bmod 26$$

$(x, y \in \mathbb{Z}_{26})$.

Consequently, $x = a^{-1}(y - b) \bmod 26$. This is an explicit formula for $x$, that is, the decryption function is

$$d_K(y) = a^{-1}(y - b) \bmod 26.$$

So, finally, the complete description of the *Affine Cipher* is given as Cryptosystem 2.3.

Let's do a small example.

***Example 2.3*** Suppose that $K = (7, 3)$. As noted above, $7^{-1} \bmod 26 = 15$. The encryption function is

$$e_K(x) = 7x + 3,$$

and the corresponding decryption function is

$$d_K(y) = 15(y - 3) = 15y - 19,$$

where all operations are performed in $\mathbb{Z}_{26}$. It is a good check to verify that $d_K(e_K(x)) = x$ for all $x \in \mathbb{Z}_{26}$. Computing in $\mathbb{Z}_{26}$, we get

$$\begin{aligned} d_K(e_K(x)) &= d_K(7x + 3) \\ &= 15(7x + 3) - 19 \\ &= x + 45 - 19 \\ &= x. \end{aligned}$$

To illustrate, let's encrypt the plaintext *hot*. We first convert the letters *h*, *o*, *t* to residues modulo 26. These are respectively 7, 14, and 19. Now, we encrypt:

$$\begin{aligned} (7 \times 7 + 3) \bmod 26 &= 52 \bmod 26 = 0 \\ (7 \times 14 + 3) \bmod 26 &= 101 \bmod 26 = 23 \\ (7 \times 19 + 3) \bmod 26 &= 136 \bmod 26 = 6. \end{aligned}$$

So the three ciphertext characters are 0, 23, and 6, which corresponds to the alphabetic string *AXG*. We leave the decryption as an exercise for the reader. ∎

**Cryptosystem 2.4:** *Vigenère Cipher*

Let $m$ be a positive integer. Define $\mathcal{P} = \mathcal{C} = \mathcal{K} = (\mathbb{Z}_{26})^m$. For a key $K = (k_1, k_2, \ldots, k_m)$, we define

$$e_K(x_1, x_2, \ldots, x_m) = (x_1 + k_1, x_2 + k_2, \ldots, x_m + k_m)$$

and

$$d_K(y_1, y_2, \ldots, y_m) = (y_1 - k_1, y_2 - k_2, \ldots, y_m - k_m),$$

where all operations are performed in $\mathbb{Z}_{26}$.

### 2.1.4 The Vigenère Cipher

In both the *Shift Cipher* and the *Substitution Cipher*, once a key is chosen, each alphabetic character is mapped to a unique alphabetic character. For this reason, these cryptosystems are called ***monoalphabetic cryptosystems***. We now present a cryptosystem that is not monoalphabetic, the well-known *Vigenère Cipher*, as Cryptosystem 2.4. This cipher is named after Blaise de Vigenère, who lived in the sixteenth century.

Using the correspondence $A \leftrightarrow 0$, $B \leftrightarrow 1, \ldots, Z \leftrightarrow 25$ described earlier, we can associate each key $K$ with an alphabetic string of length $m$, called a ***keyword***. The *Vigenère Cipher* encrypts $m$ alphabetic characters at a time: each plaintext element is equivalent to $m$ alphabetic characters.

Let's do a small example.

***Example 2.4*** Suppose $m = 6$ and the keyword is $CIPHER$. This corresponds to the numerical equivalent $K = (2, 8, 15, 7, 4, 17)$. Suppose the plaintext is the string

`thiscryptosystemisnotsecure.`

We convert the plaintext elements to residues modulo 26, write them in groups of six, and then "add" the keyword modulo 26, as follows:

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 19 | 7 | 8 | 18 | 2 | 17 | 24 | 15 | 19 | 14 | 18 | 24 |
| 2 | 8 | 15 | 7 | 4 | 17 | 2 | 8 | 15 | 7 | 4 | 17 |
| 21 | 15 | 23 | 25 | 6 | 8 | 0 | 23 | 8 | 21 | 22 | 15 |

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 18 | 19 | 4 | 12 | 8 | 18 | 13 | 14 | 19 | 18 | 4 | 2 |
| 2 | 8 | 15 | 7 | 4 | 17 | 2 | 8 | 15 | 7 | 4 | 17 |
| 20 | 1 | 19 | 19 | 12 | 9 | 15 | 22 | 8 | 25 | 8 | 19 |

| | | |
|---|---|---|
| 20 | 17 | 4 |
| 2 | 8 | 15 |
| 22 | 25 | 19 |

The alphabetic equivalent of the ciphertext string would thus be:

VPXZGIAXIVWPUBTTMJPWIZITWZT.

To decrypt, we can use the same keyword, but we would subtract it modulo 26 from the ciphertext, instead of adding it. □

Observe that the number of possible keywords of length $m$ in a *Vigenère Cipher* is $26^m$, so even for relatively small values of $m$, an exhaustive key search would require a long time. For example, if we take $m = 5$, then the keyspace has size exceeding $1.1 \times 10^7$. This is already large enough to preclude exhaustive key search by hand (but not by computer).

In a *Vigenère Cipher* having keyword length $m$, an alphabetic character can be mapped to one of $m$ possible alphabetic characters (assuming that the keyword contains $m$ distinct characters). Such a cryptosystem is called a ***polyalphabetic cryptosystem***. In general, cryptanalysis is more difficult for polyalphabetic than for monoalphabetic cryptosystems.

### 2.1.5 The Hill Cipher

In this section, we describe another polyalphabetic cryptosystem called the *Hill Cipher*. This cipher was invented in 1929 by Lester S. Hill. Let $m$ be a positive integer, and define $\mathcal{P} = \mathcal{C} = (\mathbb{Z}_{26})^m$. The idea is to take $m$ linear combinations of the $m$ alphabetic characters in one plaintext element, thus producing the $m$ alphabetic characters in one ciphertext element.

For example, if $m = 2$, we could write a plaintext element as $x = (x_1, x_2)$ and a ciphertext element as $y = (y_1, y_2)$. Here, $y_1$ would be a linear combination of $x_1$ and $x_2$, as would $y_2$. We might take

$$\begin{aligned} y_1 &= (11x_1 + 3x_2) \bmod 26 \\ y_2 &= (8x_1 + 7x_2) \bmod 26. \end{aligned}$$

Of course, this can be written more succinctly in matrix notation as follows:

$$(y_1, y_2) = (x_1, x_2) \begin{pmatrix} 11 & 8 \\ 3 & 7 \end{pmatrix},$$

where all operations are performed in $\mathbb{Z}_{26}$. In general, we will take an $m \times m$ matrix $K$ as our key. If the entry in row $i$ and column $j$ of $K$ is $k_{i,j}$, then we write $K = (k_{i,j})$. For $x = (x_1, \ldots, x_m) \in \mathcal{P}$ and $K \in \mathcal{K}$, we compute $y = e_K(x) = (y_1, \ldots, y_m)$ as follows:

$$(y_1, y_2, \ldots, y_m) = (x_1, x_2, \ldots, x_m) \begin{pmatrix} k_{1,1} & k_{1,2} & \ldots & k_{1,m} \\ k_{2,1} & k_{2,2} & \ldots & k_{2,m} \\ \vdots & \vdots & & \vdots \\ k_{m,1} & k_{m,2} & \ldots & k_{m,m} \end{pmatrix}.$$

In other words, using matrix notation, $y = xK$.

We say that the ciphertext is obtained from the plaintext by means of a ***linear transformation***. We have to consider how decryption will work, that is, how $x$ can be computed from $y$. Readers familiar with linear algebra will realize that we will use the inverse matrix $K^{-1}$ to decrypt. The ciphertext is decrypted using the matrix equation $x = yK^{-1}$.

Here are the definitions of necessary concepts from linear algebra. If $A = (a_{i,j})$ is an $\ell \times m$ matrix and $B = (b_{j,k})$ is an $m \times n$ matrix, then we define the ***matrix product*** $AB = (c_{i,k})$ by the formula

$$c_{i,k} = \sum_{j=1}^{m} a_{i,j} b_{j,k}$$

for $1 \leq i \leq \ell$ and $1 \leq k \leq n$. That is, the entry in row $i$ and column $k$ of $AB$ is formed by taking the $i$th row of $A$ and the $k$th column of $B$, multiplying corresponding entries together, and summing. Note that $AB$ is an $\ell \times n$ matrix.

Matrix multiplication is associative (that is, $(AB)C = A(BC)$) but not, in general, commutative (it is not always the case that $AB = BA$, even for square matrices $A$ and $B$).

The $m \times m$ ***identity matrix***, denoted by $I_m$, is the $m \times m$ matrix with 1's on the main diagonal and 0's elsewhere. Thus, the $2 \times 2$ identity matrix is

$$I_2 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

$I_m$ is termed an identity matrix since $AI_m = A$ for any $\ell \times m$ matrix $A$ and $I_m B = B$ for any $m \times n$ matrix $B$. Now, the ***inverse matrix*** of an $m \times m$ matrix $A$ (if it exists) is the matrix $A^{-1}$ such that $AA^{-1} = A^{-1}A = I_m$. Not all matrices have inverses, but if an inverse exists, it is unique.

With these facts at hand, it is easy to derive the decryption formula given above, assuming that $K$ has an inverse matrix $K^{-1}$. Since $y = xK$, we can multiply both sides of the formula by $K^{-1}$, obtaining

$$yK^{-1} = (xK)K^{-1} = x(KK^{-1}) = xI_m = x.$$

(Note the use of the associativity property.)

We can verify that the example encryption matrix defined above has an inverse in $\mathbb{Z}_{26}$:

$$\begin{pmatrix} 11 & 8 \\ 3 & 7 \end{pmatrix}^{-1} = \begin{pmatrix} 7 & 18 \\ 23 & 11 \end{pmatrix}$$

since

$$\begin{aligned} \begin{pmatrix} 11 & 8 \\ 3 & 7 \end{pmatrix} \begin{pmatrix} 7 & 18 \\ 23 & 11 \end{pmatrix} &= \begin{pmatrix} 11 \times 7 + 8 \times 23 & 11 \times 18 + 8 \times 11 \\ 3 \times 7 + 7 \times 23 & 3 \times 18 + 7 \times 11 \end{pmatrix} \\ &= \begin{pmatrix} 261 & 286 \\ 182 & 131 \end{pmatrix} \\ &= \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}. \end{aligned}$$

(Remember that all arithmetic operations are done modulo 26.)

Let's now do an example to illustrate encryption and decryption in the *Hill Cipher*.

***Example 2.5*** Suppose the key is

$$K = \begin{pmatrix} 11 & 8 \\ 3 & 7 \end{pmatrix}.$$

From the computations above, we have that

$$K^{-1} = \begin{pmatrix} 7 & 18 \\ 23 & 11 \end{pmatrix}.$$

Suppose we want to encrypt the plaintext *july*. We have two elements of plaintext to encrypt: $(9, 20)$ (corresponding to *ju*) and $(11, 24)$ (corresponding to *ly*). We compute as follows:

$$(9,20)\begin{pmatrix} 11 & 8 \\ 3 & 7 \end{pmatrix} = (99+60, 72+140) = (3,4)$$

and

$$(11,24)\begin{pmatrix} 11 & 8 \\ 3 & 7 \end{pmatrix} = (121+72, 88+168) = (11,22).$$

Hence, the encryption of *july* is *DELW*. To decrypt, Bob would compute:

$$(3,4)\begin{pmatrix} 7 & 18 \\ 23 & 11 \end{pmatrix} = (9,20)$$

and

$$(11,22)\begin{pmatrix} 7 & 18 \\ 23 & 11 \end{pmatrix} = (11,24).$$

Hence, the correct plaintext is obtained. □

At this point, we have shown that decryption is possible if $K$ has an inverse. In fact, for decryption to be possible, it is necessary that $K$ has an inverse. (This follows fairly easily from elementary linear algebra, but we will not give a proof here.) So we are interested precisely in those matrices $K$ that are invertible.

The invertibility of a (square) matrix depends on the value of its determinant, which we define now.

**Definition 2.5:** Suppose that $A = (a_{i,j})$ is an $m \times m$ matrix. For $1 \le i \le m$, $1 \le j \le m$, define $A_{ij}$ to be the matrix obtained from $A$ by deleting the $i$th row and the $j$th column. The ***determinant*** of $A$, denoted $\det A$, is the value $a_{1,1}$ if $m = 1$. If $m > 1$, then $\det A$ is computed recursively from the formula

$$\det A = \sum_{j=1}^{m} (-1)^{i+j} a_{i,j} \det A_{ij},$$

where $i$ is any fixed integer between 1 and $m$.

It is not at all obvious that the value of $\det A$ is independent of the choice of $i$ in the formula given above, but it can be proved that this is indeed the case. It will be useful to write out the formulas for determinants of $2 \times 2$ and $3 \times 3$ matrices. If $A = (a_{i,j})$ is a $2 \times 2$ matrix, then

$$\det A = a_{1,1}a_{2,2} - a_{1,2}a_{2,1}.$$

If $A = (a_{i,j})$ is a $3 \times 3$ matrix, then

$$\begin{aligned} \det A &= a_{1,1}a_{2,2}a_{3,3} + a_{1,2}a_{2,3}a_{3,1} + a_{1,3}a_{2,1}a_{3,2} \\ &\quad -(a_{1,1}a_{2,3}a_{3,2} + a_{1,2}a_{2,1}a_{3,3} + a_{1,3}a_{2,2}a_{3,1}). \end{aligned}$$

For large $m$, the recursive formula given in the definition above is not usually a very efficient method of computing the determinant of an $m \times m$ square matrix. A preferred method is to compute the determinant using so-called "elementary row operations"; see any text on linear algebra.

Two important properties of determinants that we will use are $\det I_m = 1$ and the multiplication rule $\det(AB) = \det A \times \det B$.

A real matrix $K$ has an inverse if and only if its determinant is non-zero. However, it is important to remember that we are working over $\mathbb{Z}_{26}$. The relevant result for our purposes is that a matrix $K$ has an inverse modulo 26 if and only if $\gcd(\det K, 26) = 1$. To see that this condition is necessary, suppose $K$ has an inverse, denoted $K^{-1}$. By the multiplication rule for determinants, we have

$$1 = \det I = \det(KK^{-1}) = \det K \det K^{-1}.$$

Hence, $\det K$ is invertible in $\mathbb{Z}_{26}$, which is true if and only if $\gcd(\det K, 26) = 1$.

Sufficiency of this condition can be established in several ways. We will give an explicit formula for the inverse of the matrix $K$. Define a matrix $K^*$ to have as its $(i,j)$-entry the value $(-1)^{i+j} \det K_{ji}$. (Recall that $K_{ji}$ is obtained from $K$ by deleting the $j$th row and the $i$th column.) $K^*$ is called the ***adjoint matrix*** of $K$. We state the following theorem, concerning inverses of matrices over $\mathbb{Z}_n$, without proof.

**THEOREM 2.3** *Suppose $K = (k_{i,j})$ is an $m \times m$ matrix over $\mathbb{Z}_n$ such that $\det K$ is invertible in $\mathbb{Z}_n$. Then $K^{-1} = (\det K)^{-1}K^*$, where $K^*$ is the adjoint matrix of $K$.*

**REMARK** The above formula for $K^{-1}$ is not very efficient computationally, except for small values of $m$ (e.g., $m = 2, 3$). For larger $m$, the preferred method of computing inverse matrices would involve performing elementary row operations on the matrix $K$. ∎

In the $2 \times 2$ case, we have the following formula, which is an immediate corollary of Theorem 2.3.

**COROLLARY 2.4** *Suppose*

$$K = \begin{pmatrix} k_{1,1} & k_{1,2} \\ k_{2,1} & k_{2,2} \end{pmatrix}$$

*is a matrix having entries in* $\mathbb{Z}_n$*, and* $\det K = k_{1,1}k_{2,2} - k_{1,2}k_{2,1}$ *is invertible in* $\mathbb{Z}_n$*. Then*

$$K^{-1} = (\det K)^{-1} \begin{pmatrix} k_{2,2} & -k_{1,2} \\ -k_{2,1} & k_{1,1} \end{pmatrix}.$$

Let's look again at the example considered earlier. First, we have

$$\begin{aligned} \det \begin{pmatrix} 11 & 8 \\ 3 & 7 \end{pmatrix} &= (11 \times 7 - 8 \times 3) \bmod 26 \\ &= (77 - 24) \bmod 26 \\ &= 53 \bmod 26 \\ &= 1. \end{aligned}$$

Now, $1^{-1} \bmod 26 = 1$, so the inverse matrix is

$$\begin{pmatrix} 11 & 8 \\ 3 & 7 \end{pmatrix}^{-1} = \begin{pmatrix} 7 & 18 \\ 23 & 11 \end{pmatrix},$$

as we verified earlier.

Here is another example, using a $3 \times 3$ matrix.

***Example 2.6*** Suppose that

$$K = \begin{pmatrix} 10 & 5 & 12 \\ 3 & 14 & 21 \\ 8 & 9 & 11 \end{pmatrix},$$

where all entries are in $\mathbb{Z}_{26}$. The reader can verify that $\det K = 7$. In $\mathbb{Z}_{26}$, we have that $7^{-1} \bmod 26 = 15$. The adjoint matrix is

$$K^* = \begin{pmatrix} 17 & 1 & 15 \\ 5 & 14 & 8 \\ 19 & 2 & 21 \end{pmatrix}.$$

Finally, the inverse matrix is

$$K^{-1} = 15K^* = \begin{pmatrix} 21 & 15 & 17 \\ 23 & 2 & 16 \\ 25 & 4 & 3 \end{pmatrix}.$$

□

As mentioned above, encryption in the *Hill Cipher* is done by multiplying the plaintext by the matrix $K$, while decryption multiplies the ciphertext by the inverse matrix $K^{-1}$. We now give a precise mathematical description of the *Hill Cipher* over $\mathbb{Z}_{26}$; see Cryptosystem 2.5.

**Cryptosystem 2.5:** *Hill Cipher*

Let $m \geq 2$ be an integer. Let $\mathcal{P} = \mathcal{C} = (\mathbb{Z}_{26})^m$ and let

$$\mathcal{K} = \{m \times m \text{ invertible matrices over } \mathbb{Z}_{26}\}.$$

For a key $K$, we define

$$e_K(x) = xK$$

and

$$d_K(y) = yK^{-1},$$

where all operations are performed in $\mathbb{Z}_{26}$.

### 2.1.6 The Permutation Cipher

All of the cryptosystems we have discussed so far involve substitution: plaintext characters are replaced by different ciphertext characters. The idea of a permutation cipher is to keep the plaintext characters unchanged, but to alter their positions by rearranging them using a permutation.

A ***permutation*** of a finite set $X$ is a bijective function $\pi : X \to X$. In other words, the function $\pi$ is one-to-one (***injective***) and onto (***surjective***). It follows that, for every $x \in X$, there is a unique element $x' \in X$ such that $\pi(x') = x$. This allows us to define the ***inverse permutation***, $\pi^{-1} : X \to X$ by the rule

$$\pi^{-1}(x) = x' \quad \text{if and only if} \quad \pi(x') = x.$$

Then $\pi^{-1}$ is also a permutation of $X$.

The *Permutation Cipher* (also known as the *Transposition Cipher*) is defined formally as Cryptosystem 2.6. This cryptosystem has been in use for hundreds of years. In fact, the distinction between the *Permutation Cipher* and the *Substitution Cipher* was pointed out as early as 1563 by Giovanni Porta.

As with the *Substitution Cipher*, it is more convenient to use alphabetic characters as opposed to residues modulo 26, since there are no algebraic operations being performed in encryption or decryption.

Here is an example to illustrate:

***Example 2.7*** Suppose $m = 6$ and the key is the following permutation $\pi$:

| $x$ | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| $\pi(x)$ | 3 | 5 | 1 | 6 | 4 | 2 |

Note that the first row of the above diagram lists the values of $x$, $1 \leq x \leq 6$, and the second row lists the corresponding values of $\pi(x)$. Then the inverse permutation $\pi^{-1}$ can be constructed by interchanging the two rows, and rearranging the

**Cryptosystem 2.6:** *Permutation Cipher*

Let $m$ be a positive integer. Let $\mathcal{P} = \mathcal{C} = (\mathbb{Z}_{26})^m$ and let $\mathcal{K}$ consist of all permutations of $\{1, \ldots, m\}$. For a key (i.e., a permutation) $\pi$, we define

$$e_\pi(x_1, \ldots, x_m) = (x_{\pi(1)}, \ldots, x_{\pi(m)})$$

and

$$d_\pi(y_1, \ldots, y_m) = (y_{\pi^{-1}(1)}, \ldots, y_{\pi^{-1}(m)}),$$

where $\pi^{-1}$ is the inverse permutation to $\pi$.

columns so that the first row is in increasing order. Carrying out these operations, we see that the permutation $\pi^{-1}$ is the following:

| $x$ | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| $\pi^{-1}(x)$ | 3 | 6 | 1 | 5 | 2 | 4 |

Now, suppose we are given the plaintext

`shesellsseashellsbytheseashore`.

We first partition the plaintext into groups of six letters:

`shesel` | `lsseas` | `hellsb` | `ythese` | `ashore`

Now each group of six letters is rearranged according to the permutation $\pi$, yielding the following:

`EESLSH` | `SALSES` | `LSHBLE` | `HSYEET` | `HRAEOS`

So, the ciphertext is:

`EESLSHSALSESLSHBLEHSYEETHRAEOS`.

The ciphertext can be decrypted in a similar fashion, using the inverse permutation $\pi^{-1}$. □

We now show that the *Permutation Cipher* is a special case of the *Hill Cipher*. Given a permutation $\pi$ of the set $\{1, \ldots, m\}$, we can define an associated $m \times m$ permutation matrix $K_\pi = (k_{i,j})$ according to the formula

$$k_{i,j} = \begin{cases} 1 & \text{if } i = \pi(j) \\ 0 & \text{otherwise.} \end{cases}$$

(A ***permutation matrix*** is a matrix in which every row and column contains exactly

one "1," and all other values are "0." A permutation matrix can be obtained from an identity matrix by permuting rows or columns.)

It is not difficult to see that Hill encryption using the matrix $K_\pi$ is, in fact, equivalent to permutation encryption using the permutation $\pi$. Moreover, ${K_\pi}^{-1} = K_{\pi^{-1}}$, i.e., the inverse matrix to $K_\pi$ is the permutation matrix defined by the permutation $\pi^{-1}$. Thus, Hill decryption is equivalent to permutation decryption.

For the permutation $\pi$ used in the example above, the associated permutation matrices are

$$K_\pi = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

and

$${K_\pi}^{-1} = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

The reader can verify that the product of these two matrices is the identity matrix.

### 2.1.7 Stream Ciphers

In the cryptosystems we have studied so far, successive plaintext elements are encrypted using the same key, $K$. That is, the ciphertext string $\mathbf{y}$ is obtained as follows:

$$\mathbf{y} = y_1 y_2 \cdots = e_K(x_1) e_K(x_2) \cdots .$$

Cryptosystems of this type are often called ***block ciphers***.

An alternative approach is to use what are called stream ciphers. The basic idea is to generate a keystream $\mathbf{z} = z_1 z_2 \cdots$, and use it to encrypt a plaintext string $\mathbf{x} = x_1 x_2 \cdots$ according to the rule

$$\mathbf{y} = y_1 y_2 \cdots = e_{z_1}(x_1) e_{z_2}(x_2) \cdots .$$

The simplest type of stream cipher is one in which the keystream is constructed from the key, independent of the plaintext string, using some specified algorithm. This type of stream cipher is called "synchronous" and can be defined formally as follows:

**Definition 2.6:** A ***synchronous stream cipher*** is a tuple $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{L}, \mathcal{E}, \mathcal{D})$, together with a function $g$, such that the following conditions are satisfied:

1. $\mathcal{P}$ is a finite set of possible plaintexts
2. $\mathcal{C}$ is a finite set of possible ciphertexts
3. $\mathcal{K}$, the keyspace, is a finite set of possible keys
4. $\mathcal{L}$ is a finite set called the ***keystream alphabet***
5. $g$ is the ***keystream generator***. $g$ takes a key $K$ as input, and generates an infinite string $z_1 z_2 \cdots$ called the ***keystream***, where $z_i \in \mathcal{L}$ for all $i \geq 1$.
6. For each $z \in \mathcal{L}$, there is an encryption rule $e_z \in \mathcal{E}$ and a corresponding decryption rule $d_z \in \mathcal{D}$. $e_z : \mathcal{P} \to \mathcal{C}$ and $d_z : \mathcal{C} \to \mathcal{P}$ are functions such that $d_z(e_z(x)) = x$ for every plaintext element $x \in \mathcal{P}$.

To illustrate this definition, we show how the *Vigenère Cipher* can be defined as a synchronous stream cipher. Suppose that $m$ is the keyword length of a *Vigenère Cipher*. Define $\mathcal{K} = (\mathbb{Z}_{26})^m$ and $\mathcal{P} = \mathcal{C} = \mathcal{L} = \mathbb{Z}_{26}$; and define $e_z(x) = (x + z) \bmod 26$ and $d_z(y) = (y - z) \bmod 26$. Finally, define the keystream $z_1 z_2 \cdots$ as follows:

$$z_i = \begin{cases} k_i & \text{if } 1 \leq i \leq m \\ z_{i-m} & \text{if } i \geq m+1, \end{cases}$$

where $K = (k_1, \ldots, k_m)$. This generates the keystream

$$k_1 k_2 \cdots k_m k_1 k_2 \cdots k_m k_1 k_2 \cdots$$

from the key $K = (k_1, k_2, \ldots, k_m)$.

**REMARK** We can think of a block cipher as a special case of a stream cipher where the keystream is constant: $z_i = K$ for all $i \geq 1$. ∎

A stream cipher is a ***periodic stream cipher*** with period $d$ if $z_{i+d} = z_i$ for all integers $i \geq 1$. The *Vigenère Cipher* with keyword length $m$, as described above, can be thought of as a periodic stream cipher with period $m$.

Stream ciphers are often described in terms of binary alphabets, i.e., $\mathcal{P} = \mathcal{C} = \mathcal{L} = \mathbb{Z}_2$. In this situation, the encryption and decryption operations are just addition modulo 2:

$$e_z(x) = (x + z) \bmod 2$$

and

$$d_z(y) = (y + z) \bmod 2.$$

If we think of "0" as representing the boolean value "false" and "1" as representing

"true," then addition modulo 2 corresponds to the ***exclusive-or*** operation. Hence, encryption (and decryption) can be implemented very efficiently in hardware.

Let's look at another method of generating a (synchronous) keystream. We will work over binary alphabets. Suppose we start with a binary $m$-tuple $(k_1, \ldots, k_m)$ and let $z_i = k_i$, $1 \leq i \leq m$ (as before). Now we generate the keystream using a ***linear recurrence*** of degree $m$:

$$z_{i+m} = \sum_{j=0}^{m-1} c_j z_{i+j} \bmod 2,$$

for all $i \geq 1$, where $c_0, \ldots, c_{m-1} \in \mathbb{Z}_2$ are specified constants.

**REMARK** This recurrence is said to have ***degree*** $m$ since each term depends on the previous $m$ terms. It is a ***linear recurrence*** because $z_{i+m}$ is a linear function of previous terms. Note that we can take $c_0 = 1$ without loss of generality, for otherwise the recurrence will be of degree (at most) $m - 1$. ∎

Here, the key $K$ consists of the $2m$ values $k_1, \ldots, k_m, c_0, \ldots, c_{m-1}$. If

$$(k_1, \ldots, k_m) = (0, \ldots, 0),$$

then the keystream consists entirely of 0's. Of course, this should be avoided, as the ciphertext will then be identical to the plaintext. However, if the constants $c_0, \ldots, c_{m-1}$ are chosen in a suitable way, then any other initialization vector $(k_1, \ldots, k_m)$ will give rise to a periodic keystream having period $2^m - 1$. So a "short" key can give rise to a keystream having a very long period. This is certainly a desirable property: we will see in a later section how the *Vigenère Cipher* can be cryptanalyzed by exploiting the fact that the keystream has a short period.

Here is an example to illustrate.

***Example 2.8*** Suppose $m = 4$ and the keystream is generated using the linear recurrence

$$z_{i+4} = (z_i + z_{i+1}) \bmod 2,$$

$i \geq 1$. If the keystream is initialized with any vector other than $(0, 0, 0, 0)$, then we obtain a keystream of period 15. For example, starting with $(1, 0, 0, 0)$, the keystream is

$$1\,0\,0\,0\,1\,0\,0\,1\,1\,0\,1\,0\,1\,1\,1 \cdots .$$

Any other non-zero initialization vector will give rise to a cyclic permutation of the same keystream. □

Another appealing aspect of this method of keystream generation is that the keystream can be produced efficiently in hardware using a ***linear feedback shift register***, or ***LFSR***. We would use a shift register with $m$ ***stages***. The vector $(k_1, \ldots, k_m)$ would be used to initialize the shift register. At each time unit, the following operations would be performed concurrently:

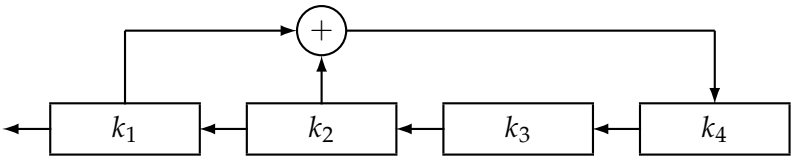


**FIGURE 2.2**: A linear feedback shift register

1. $k_1$ would be tapped as the next keystream bit
2. $k_2, \ldots, k_m$ would each be shifted one stage to the left
3. the "new" value of $k_m$ would be computed to be

$$\sum_{j=0}^{m-1} c_j k_{j+1}$$

(this is the "linear feedback").

At any given point in time, the shift register contains $m$ consecutive keystream elements, say $z_i, \ldots, z_{i+m-1}$. After one time unit, the shift register contains $z_{i+1}, \ldots, z_{i+m}$.

Observe that the linear feedback is carried out by tapping certain stages of the register (as specified by the constants $c_j$ having the value "1") and computing a sum modulo 2 (which is an exclusive-or). This is illustrated in Figure 2.2, where we depict the LFSR that will generate the keystream of Example 2.8.

A ***non-synchronous stream cipher*** is a stream cipher in which each keystream element $z_i$ depends on previous plaintext or ciphertext elements ($x_1, \ldots, x_{i-1}$ and/or $y_1, \ldots, y_{i-1}$) as well as the key $K$. A simple type of non-synchronous stream cipher, known as the *Autokey Cipher*, is presented as Cryptosystem 2.7. It is apparently due to Vigenère. The reason for the terminology "autokey" is that the plaintext is used to construct the keystream (aside from the initial "priming key" $K$). Of course, the *Autokey Cipher* is insecure since there are only 26 possible keys.

Here is an example to illustrate:

***Example 2.9*** Suppose the key is $K = 8$, and the plaintext is

`rendezvous.`

We first convert the plaintext to a sequence of integers:

17 4 13 3 4 25 21 14 20 18

The keystream is as follows:

8 17 4 13 3 4 25 21 14 20

**Cryptosystem 2.7:** *Autokey Cipher*

Let $\mathcal{P} = \mathcal{C} = \mathcal{K} = \mathcal{L} = \mathbb{Z}_{26}$. Let $z_1 = K$, and define $z_i = x_{i-1}$ for all $i \geq 2$. For $0 \leq z \leq 25$, define

$$e_z(x) = (x + z) \bmod 26$$

and

$$d_z(y) = (y - z) \bmod 26$$

$(x, y \in \mathbb{Z}_{26})$.

Now we add corresponding elements, reducing modulo 26:

$$25 \quad 21 \quad 17 \quad 16 \quad 7 \quad 3 \quad 20 \quad 9 \quad 8 \quad 12$$

In alphabetic form, the ciphertext is:

ZVRQHDUJIM.

Now let's look at how the ciphertext would be decrypted. First, we convert the alphabetic string to the numeric string

$$25 \quad 21 \quad 17 \quad 16 \quad 7 \quad 3 \quad 20 \quad 9 \quad 8 \quad 12$$

Then we compute

$$x_1 = d_8(25) = (25 - 8) \bmod 26 = 17.$$

Next,

$$x_2 = d_{17}(21) = (21 - 17) \bmod 26 = 4,$$

and so on. Each time we obtain another plaintext character, we also use it as the next keystream element. □

In the next section, we discuss methods that can be used to cryptanalyze the various cryptosystems we have presented.

## 2.2 Cryptanalysis

In this section, we discuss some techniques of cryptanalysis. The general assumption that is usually made is that the opponent, Oscar, knows the cryptosystem being used. This is usually referred to as ***Kerckhoffs' Principle***. Of course, if Oscar does not know the cryptosystem being used, that will make his task more

difficult. But we do not want to base the security of a cryptosystem on the (possibly shaky) premise that Oscar does not know what system is being employed. Hence, our goal in designing a cryptosystem will be to obtain security while assuming that Kerckhoffs' principle holds.

First, we want to differentiate between different attack models on cryptosystems. The ***attack model*** specifies the information available to the adversary when he mounts his attack. The most common types of attack models are enumerated as follows.

**ciphertext-only attack**
: The opponent possesses a string of ciphertext, **y**.

**known plaintext attack**
: The opponent possesses a string of plaintext, **x**, and the corresponding ciphertext, **y**.

**chosen plaintext attack**
: The opponent has obtained temporary access to the encryption machinery. Hence he can choose a plaintext string, **x**, and construct the corresponding ciphertext string, **y**.

**chosen ciphertext attack**
: The opponent has obtained temporary access to the decryption machinery. Hence he can choose a ciphertext string, **y**, and construct the corresponding plaintext string, **x**.

In each case, the objective of the adversary is to determine the key that was used. This would allow the opponent to decrypt a specific "target" ciphertext string, and further, to decrypt any additional ciphertext strings that are encrypted using the same key.

At first glance, a chosen ciphertext attack may seem to be a bit artificial. For, if there is only one ciphertext string of interest to the opponent, then the opponent can obviously decrypt that ciphertext string if a chosen ciphertext attack is permitted. However, we are suggesting that the opponent's objective normally includes determining the key that is used by Alice and Bob, so that other ciphertext strings can be decrypted (at a later time, perhaps). A chosen ciphertext attack makes sense in this context.

We first consider the weakest type of attack, namely a ciphertext-only attack (this is sometimes called a ***known ciphertext attack***). We also assume that the plaintext string is ordinary English text, without punctuation or "spaces." (This makes cryptanalysis more difficult than if punctuation and spaces were encrypted.)

Many techniques of cryptanalysis use statistical properties of the English language. Various people have estimated the relative frequencies of the 26 letters by compiling statistics from numerous novels, magazines, and newspapers. The estimates in Table 2.1 were obtained by Beker and Piper. On the basis of these probabilities, Beker and Piper partition the 26 letters into five groups as follows:

**TABLE 2.1**: Probabilities of occurrence of the 26 letters

| letter | probability | letter | probability |
|---|---|---|---|
| $A$ | .082 | $N$ | .067 |
| $B$ | .015 | $O$ | .075 |
| $C$ | .028 | $P$ | .019 |
| $D$ | .043 | $Q$ | .001 |
| $E$ | .127 | $R$ | .060 |
| $F$ | .022 | $S$ | .063 |
| $G$ | .020 | $T$ | .091 |
| $H$ | .061 | $U$ | .028 |
| $I$ | .070 | $V$ | .010 |
| $J$ | .002 | $W$ | .023 |
| $K$ | .008 | $X$ | .001 |
| $L$ | .040 | $Y$ | .020 |
| $M$ | .024 | $Z$ | .001 |

1. $E$, having probability about 0.120
2. $T, A, O, I, N, S, H, R$, each having probability between 0.06 and 0.09
3. $D, L$, each having probability around 0.04
4. $C, U, M, W, F, G, Y, P, B$, each having probability between 0.015 and 0.028
5. $V, K, J, X, Q, Z$, each having probability less than 0.01.

It is also useful to consider sequences of two or three consecutive letters, called ***digrams*** and ***trigrams***, respectively. The 30 most common digrams are (in decreasing order):

$$TH, HE, IN, ER, AN, RE, ED, ON, ES, ST,$$
$$EN, AT, TO, NT, HA, ND, OU, EA, NG, AS,$$
$$OR, TI, IS, ET, IT, AR, TE, SE, HI, OF.$$

The twelve most common trigrams are:

$$THE, ING, AND, HER, ERE, ENT,$$
$$THA, NTH, WAS, ETH, FOR, DTH.$$

### 2.2.1 Cryptanalysis of the Affine Cipher

As a simple illustration of how cryptanalysis can be performed using statistical data, let's look first at the *Affine Cipher*. Suppose Oscar has intercepted the ciphertext shown in the following example:

**TABLE 2.2**: Frequency of occurrence of the 26 ciphertext letters

| letter | frequency | letter | frequency |
|---|---|---|---|
| *A* | 2 | *N* | 1 |
| *B* | 1 | *O* | 1 |
| *C* | 0 | *P* | 2 |
| *D* | 7 | *Q* | 0 |
| *E* | 5 | *R* | 8 |
| *F* | 4 | *S* | 3 |
| *G* | 0 | *T* | 0 |
| *H* | 5 | *U* | 2 |
| *I* | 0 | *V* | 4 |
| *J* | 0 | *W* | 0 |
| *K* | 5 | *X* | 2 |
| *L* | 2 | *Y* | 1 |
| *M* | 2 | *Z* | 0 |

***Example 2.10*** Ciphertext obtained from an *Affine Cipher*

```
FMXVEDKAPHFERBNDKRXRSREFMORUDSDKDVSHVUFEDK
APRKDLYEVLRHHRH
```

The frequency analysis of this ciphertext is given in Table 2.2.

There are only 57 characters of ciphertext, but this is usually sufficient to cryptanalyze an *Affine Cipher*. The most frequent ciphertext characters are: $R$ (8 occurrences), $D$ (7 occurrences), $E$, $H$, $K$ (5 occurrences each), and $F$, $S$, $V$ (4 occurrences each). As a first guess, we might hypothesize that $R$ is the encryption of $e$ and $D$ is the encryption of $t$, since $e$ and $t$ are (respectively) the two most common letters. Expressed numerically, we have $e_K(4) = 17$ and $e_K(19) = 3$. Recall that $e_K(x) = ax + b$, where $a$ and $b$ are unknowns. So we get two linear equations in two unknowns:

$$\begin{aligned} 4a + b &= 17 \\ 19a + b &= 3. \end{aligned}$$

This system has the unique solution $a = 6$, $b = 19$ (in $\mathbb{Z}_{26}$). But this is an illegal key, since $\gcd(a, 26) = 2 > 1$. So our hypothesis must be incorrect.

Our next guess might be that $R$ is the encryption of $e$ and $E$ is the encryption of $t$. Proceeding as above, we obtain $a = 13$, which is again illegal. So we try the next possibility, that $R$ is the encryption of $e$ and $H$ is the encryption of $t$. This yields $a = 8$, again impossible. Continuing, we suppose that $R$ is the encryption of $e$ and $K$ is the encryption of $t$. This produces $a = 3$, $b = 5$, which is at least a legal key. It remains to compute the decryption function corresponding to $K = (3, 5)$, and then to decrypt the ciphertext to see if we get a meaningful string of English, or nonsense. This will confirm the validity of $(3, 5)$.