密码导论及编码原理

（影印版）

Introduction to Cryptography with Coding Theory

（美） Wade Trappe
Lawrence C. Washington 编著

科学出版社

北京

图字：01-2003-6693 号

内 容 简 介

本书的一个重要特色——它还覆盖的范围相当广泛。本书前8章涵盖了密码学的主要领域：经典密码理论、基础数论、RSA算法与数字签名等。本书后9章讲述了密钥共享方案、游戏理论、信息论、电子商务、数字钞票与“零知识”技术等等。在附录部分，作者给出了分别用Mathematica、Maple和MATLAB实现算法的事例。

本书可作为高年级本科生和研究生的有关密码学与网络安全及相关专业的教材。

ISBN 0-13-061814-4

图书在版编目(CIP)数据

密码导论及编码原理=Introduction to Cryptography with Coding Theory / (美) Trappe, W. 等著. —影印本. —北京：科学出版社，2004

ISBN 7-03-012470-7

Ⅰ. 密… Ⅱ. T… Ⅲ. ①密码—理论—英文②保密编码—编码理论—英文 Ⅳ. TN918

中国版本图书馆 CIP 数据核字（2003）第 099956 号

策划编辑：李佩乾/责任编辑：朱凤成
责任印制：吕春珉/封面制作：北新华文平面设计室

科学出版社 出版
北京东黄城根北街16号
邮政编码:100717
http://www.sciencep.com

双青印刷厂 印刷

科学出版社发行 各地新华书店经销

*

2004 年 1 月第 一 版 开本：787×960 1/16
2004 年 1 月第一次印刷 印张：31 1/2
印数：1—3 000 字数：504 000

定价：53.00 元

（如有印装质量问题，我社负责调换〈环伟〉）

Contents

Preface vii

1 Overview 1
- 1.1 Secure Communications . . . 2
- 1.2 Cryptographic Applications . . . 9

2 Classical Cryptosystems 12
- 2.1 Shift Ciphers . . . 13
- 2.2 Affine Ciphers . . . 14
- 2.3 The Vigenère Cipher . . . 16
- 2.4 Substitution Ciphers . . . 23
- 2.5 Sherlock Holmes . . . 26
- 2.6 The Playfair and ADFGX Ciphers . . . 29
- 2.7 Block Ciphers . . . 33
- 2.8 Binary Numbers and ASCII . . . 37
- 2.9 One-Time Pads . . . 38
- 2.10 Pseudo-random Bit Generation . . . 40
- 2.11 Linear Feedback Shift Register Sequences . . . 42
- 2.12 Enigma . . . 49
- 2.13 Exercises . . . 54
- 2.14 Computer Problems . . . 56

3 Basic Number Theory 59
- 3.1 Basic Notions . . . 59
- 3.2 Solving $ax + by = d$ . . . 65
- 3.3 Congruences . . . 66
- 3.4 The Chinese Remainder Theorem . . . 72
- 3.5 Modular Exponentiation . . . 74
- 3.6 Fermat and Euler . . . 75
- 3.7 Primitive Roots . . . 79
- 3.8 Inverting Matrices Mod n . . . 80
- 3.9 Square Roots Mod n . . . 81

3.10 Finite Fields 83
3.11 Exercises 91
3.12 Computer Problems 95

4 The Data Encryption Standard **97**
4.1 Introduction 97
4.2 A Simplified DES-Type Algorithm 98
4.3 Differential Cryptanalysis 102
4.4 DES 107
4.5 Modes of Operation 115
4.6 Breaking DES 118
4.7 Password Security 123
4.8 Exercises 125

5 AES: Rijndael **127**
5.1 The Basic Algorithm 128
5.2 The Layers 129
5.3 Decryption 133
5.4 Design Considerations 136

6 The RSA Algorithm **137**
6.1 The RSA Algorithm 137
6.2 Attacks on RSA 142
6.3 Primality Testing 145
6.4 Factoring 149
6.5 The RSA Challenge 154
6.6 An Application to Treaty Verification 156
6.7 The Public Key Concept 156
6.8 Exercises 159
6.9 Computer Problems 162

7 Discrete Logarithms **165**
7.1 Discrete Logarithms 165
7.2 Computing Discrete Logs 166
7.3 Bit Commitment 173
7.4 The ElGamal Public Key Cryptosystem 173
7.5 Exercises 175
7.6 Computer Problems 176

8 Digital Signatures **177**
8.1 RSA Signatures 178
8.2 The ElGamal Signature Scheme 179
8.3 Hash Functions 182

8.4 Birthday Attacks . . . 186
8.5 The Digital Signature Algorithm . . . 190
8.6 Exercises . . . 191
8.7 Computer Problems . . . 194

9 E-Commerce and Digital Cash **196**
9.1 Secure Electronic Transaction . . . 197
9.2 Digital Cash . . . 199
9.3 Exercises . . . 206

10 Secret Sharing Schemes **208**
10.1 Secret Splitting . . . 208
10.2 Threshold Schemes . . . 209
10.3 Exercises . . . 215
10.4 Computer Problems . . . 217

11 Games **219**
11.1 Flipping Coins over the Telephone . . . 219
11.2 Poker over the Telephone . . . 221
11.3 Exercises . . . 226

12 Zero-Knowledge Techniques **228**
12.1 The Basic Setup . . . 228
12.2 Feige-Fiat-Shamir Identification Scheme . . . 231
12.3 Exercises . . . 233

13 Key Establishment Protocols **236**
13.1 Key Agreement Protocols . . . 237
13.2 Key Pre-distribution . . . 239
13.3 Key Distribution . . . 241
13.4 Public Key Infrastructures (PKI) . . . 246
13.5 Exercises . . . 248

14 Information Theory **250**
14.1 Probability Review . . . 251
14.2 Entropy . . . 253
14.3 Huffman Codes . . . 258
14.4 Perfect Secrecy . . . 260
14.5 The Entropy of English . . . 263
14.6 Exercises . . . 268

15 Elliptic Curves **272**
15.1 The Addition Law . . . 272
15.2 Elliptic Curves Mod n . . . 276
15.3 Factoring with Elliptic Curves . . . 280
15.4 Elliptic Curves in Characteristic 2 . . . 284
15.5 Elliptic Curve Cryptosystems . . . 287
15.6 Exercises . . . 290
15.7 Computer Problems . . . 293

16 Error Correcting Codes **295**
16.1 Introduction . . . 295
16.2 Error Correcting Codes . . . 301
16.3 Bounds on General Codes . . . 305
16.4 Linear Codes . . . 311
16.5 Hamming Codes . . . 319
16.6 Golay Codes . . . 320
16.7 Cyclic Codes . . . 329
16.8 BCH Codes . . . 335
16.9 Reed-Solomon Codes . . . 343
16.10 The McEliece Cryptosystem . . . 345
16.11 Other Topics . . . 348
16.12 Exercises . . . 349
16.13 Computer Problems . . . 352

17 Quantum Cryptography **353**
17.1 A Quantum Experiment . . . 354
17.2 Quantum Key Distribution . . . 357
17.3 Shor's Algorithm . . . 359
17.4 Exercises . . . 370

A Mathematica **372**
A.1 Getting Started with Mathematica . . . 372
A.2 Some Commands . . . 374
A.3 Examples for Chapter 2 . . . 375
A.4 Examples for Chapter 3 . . . 382
A.5 Examples for Chapter 6 . . . 386
A.6 Examples for Chapter 8 . . . 394
A.7 Examples for Chapter 10 . . . 395
A.8 Examples for Chapter 11 . . . 396
A.9 Examples for Chapter 15 . . . 397

B Maple Examples **403**
B.1 Getting Started with Maple 403
B.2 Some Commands 404
B.3 Examples for Chapter 2 406
B.4 Examples for Chapter 3 414
B.5 Examples for Chapter 6 419
B.6 Examples for Chapter 8 428
B.7 Examples for Chapter 10 428
B.8 Examples for Chapter 11 430
B.9 Examples for Chapter 15 432

C MATLAB Examples **437**
C.1 Getting Started with MATLAB 438
C.2 Examples for Chapter 2 444
C.3 Examples for Chapter 3 456
C.4 Examples for Chapter 6 460
C.5 Examples for Chapter 8 466
C.6 Examples for Chapter 10 466
C.7 Examples for Chapter 11 467
C.8 Examples for Chapter 15 470

D Further Reading **478**

Bibliography **479**

Index **485**

Preface

This book is based on a course in cryptography at the upper level undergraduate and beginning graduate level that has been given at the University of Maryland since 1997. When designing the course, we decided on the following requirements.

- The course should be up-to-date and cover a broad selection of topics from a mathematical point of view.
- The material should be accessible to mathematically mature students having little background in number theory and computer programming.
- There should be examples involving numbers large enough to demonstrate how the algorithms really work.

We wanted to avoid concentrating solely on RSA and discrete logarithms, which would have made the course mostly a number theory course. We also did not want to teach a course on protocols and how to hack into friends' computers. That would have made the course less mathematical than desired.

There are numerous topics in cryptology that can be discussed in an introductory course. We have tried to include many of them. The chapters represent, for the most part, topics that were covered during the different semesters we taught the course. There is certainly more material here than could be treated in most one-semester courses. The first eight chapters represent the core of the material. The choice of which of the remaining chapters are used depends on the level of the students.

The chapters are numbered, thus giving them an ordering. However, except for Chapter 3 on number theory, which pervades the subject, the

chapters are fairly independent of each other and can be covered in almost any reasonable order. Although we don't recommend doing so, a daring reader could possibly read Chapters 4 through 17 in reverse order, with only having to look ahead/behind a few times.

The chapters on Information Theory, Elliptic Curves, Quantum Methods, and Error Correcting Codes are somewhat more mathematical than the others. The chapter on Error Correcting Codes was included, at the suggestion of several reviewers, because courses that include introductions to both cryptology and coding theory are fairly common.

Computer examples. Suppose you want to give an example for RSA. You could choose two one-digit primes and pretend to be working with fifty-digit primes, or you could use your favorite software package to do an actual example with large primes. Or perhaps you are working with shift ciphers and are trying to decrypt a message by trying all 26 shifts of the ciphertext. This should also be done on a computer. At the end of the book are appendices containing Computer Examples written in each of Mathematica®, Maple®, and MATLAB® that show how to do such calculations. These languages were chosen because they are user friendly and do not require prior programming experience. Although the course has been taught successfully without computers, these examples are an integral part of the book and should be studied, if at all possible. Not only do they contain numerical examples of how to do certain computations but also they demonstrate important ideas and issues that arise. They were placed at the end of the book because of the logistic and aesthetic problems of including extensive computer examples in three languages at the ends of chapters.

Programs available in each of the three languages can be downloaded from the Web site

www.prenhall.com/washington

In a classroom, all that is needed is a computer (with one of the languages installed) and a projector in order to produce meaningful examples as the lecture is being given. Homework problems (the Computer Problems in various chapters) based on the software allow students to play with examples individually. Of course, students having more programming background could write their own programs instead.

Acknowledgments. Many people helped and provided encouragement during the preparation of this book. First, we would like to thank our students, whose enthusiasm, insights, and suggestions contributed greatly. We are especially grateful to David Bindel, Jason Ernst, Christine Planchak, Haw-ren Fang, Marwan Oweis, Bob Grafton, who provided many corrections and other input. Our colleague Bill Gasarch taught out of the

penultimate version of the text. His many excellent comments were extremely helpful. Jonathan Rosenberg and Tim Strobell provided invaluable technical assistance. The reviewers deserve special thanks: David Grant (University of Colorado at Boulder), David M. Pozar (University of Massachusetts, Amherst), Jugal K. Kalita (University of Colorado at Colorado Springs), Anthony Ephremides (University of Maryland, College Park), J. Felipe Voloch (University of Texas at Austin), Agnes Chan (Northeastern University), Daniel F. Warren (Naval Postgraduate School), and one anonymous reviewer. Their suggestions on the exposition and the organization of the topics greatly enhanced the final result. We have enjoyed working with the staff at Prentice Hall, especially the mathematics editor, George Lobell, and the production editor, Jeanne Audino.

The first author would like to thank Nisha Gilra, who provided encouragement and advice; Sheilagh O'Hare for introducing him to the field of cryptography; and K.J. Ray Liu for his support.

The second author thanks Susan Zengerle and Patrick Washington for their patience, help, and encouragement during the writing of this book.

Wade Trappe
wxt@math.umd.edu

Lawrence C. Washington
lcw@math.umd.edu

Chapter 1

Overview of Cryptography and Its Applications

People have always had a fascination with keeping information away from others. As children, many of us had magic decoder rings for exchanging coded messages with our friends and possibly keeping secrets from parents, siblings, or teachers. History is filled with examples where people tried to keep information secret from adversaries. Kings and generals communicated with their troops using basic cryptographic methods to prevent the enemy from learning sensitive military information. In fact, Julius Caesar reportedly used a simple cipher, which has been named after him.

As society has evolved, the need for more sophisticated methods of protecting data has increased. Now, with the information era at hand, the need is more pronounced than ever. As the world becomes more connected, the demand for information and electronic services is growing, and with the increased demand comes increased dependency on electronic systems. Already the exchange of sensitive information, such as credit card numbers, over the Internet is common practice. Protecting data and electronic systems is crucial to our way of living.

The techniques needed to protect data belong to the field of cryptography. Actually, the subject has three names, **cryptography**, **cryptology**, and **cryptanalysis**, which are often used interchangeably. Technically, however, cryptology is the all-inclusive term for the study of communication over nonsecure channels, and related problems. The process of designing systems

to do this is called cryptography. Cryptanalysis deals with breaking such systems. Of course, it is essentially impossible to do either cryptography or cryptanalysis without having a good understanding of the methods of both areas.

Often the term **coding theory** is used to describe cryptography; however, this can lead to confusion. Coding theory deals with representing input information symbols by output symbols called code symbols. There are three basic applications that coding theory covers: compression, secrecy, and error correction. Over the past few decades, the term coding theory has become associated predominantly with error correcting codes. Coding theory thus studies communication over noisy channels and how to ensure that the message received is the correct message, as opposed to cryptography, which protects communication over nonsecure channels.

Although error correcting codes are only a secondary focus of this book, we should emphasize that, in any real-world system, error correcting codes are used in conjunction with encryption, since the change of a single bit is enough to destroy the message completely in a well-designed cryptosystem.

Modern cryptography is a field that draws heavily upon mathematics, computer science, and cleverness. This book provides an introduction to the mathematics and protocols needed to make data transmission and electronic systems secure, along with techniques such as electronic signatures and secret sharing.

1.1 Secure Communications

In the basic communication scenario, depicted in Figure 1.1, there are two parties, we'll call them Alice and Bob, who want to communicate with each other. A third party, Eve, is a potential eavesdropper.

When Alice wants to send a message, called the **plaintext**, to Bob, she encrypts it using a method prearranged with Bob. Usually, the encryption method is assumed to be known to Eve; what keeps the message secret is a **key**. When Bob receives the encrypted message, called the **ciphertext**, he changes it back to the plaintext using a decryption key.

Eve could have one of the following goals:

1. Read the message.
2. Find the key and thus read all messages encrypted with that key.
3. Corrupt Alice's message into another message in such a way that Bob will think Alice sent the altered message.
4. Masquerade as Alice, and thus communicate with Bob even though Bob believes he is communicating with Alice.

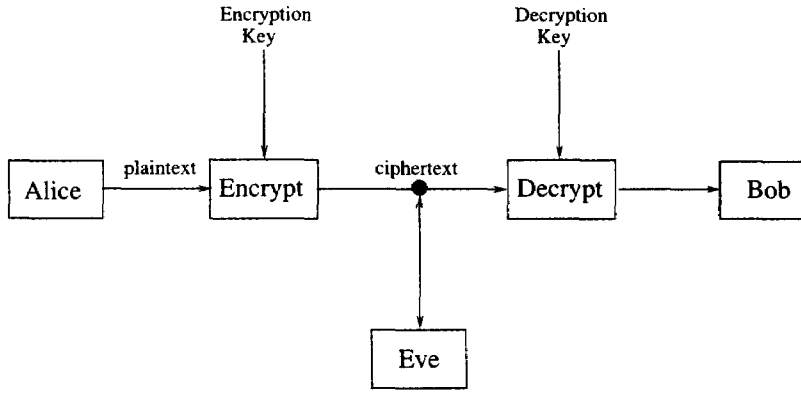

Figure 1.1: The Basic Communication Scenario for Cryptography.

Which case we're in depends on how evil Eve is. Cases (3) and (4) relate to issues of integrity and authentication, respectively. We'll discuss these shortly. A more active and malicious adversary, corresponding to cases (3) and (4), is sometimes called Mallory in the literature. More passive observers (as in cases (1) and (2)) are sometimes named Oscar. We'll generally use only Eve, and assume she is as bad as the situation allows.

Possible Attacks

There are four main types of attack that Eve might be able to use. The differences among these types of attacks are the amounts of information Eve has available to her when trying to determine the key. The four attacks are as follows:

1. **Ciphertext only:** Eve has only a copy of the ciphertext.

2. **Known plaintext:** Eve has a copy of a ciphertext and the corresponding plaintext. For example, suppose Eve intercepts an encrypted press release, then sees the decrypted release the next day. If she can deduce the decryption key, and if Alice doesn't change the key, Eve can read all future messages. Or, if Alice always starts her messages with "Dear Bob," then Eve has a small piece of ciphertext and corresponding plaintext. For many weak cryptosystems, this suffices to find the key. Even for stronger systems such as the German Enigma machine used in World War II, this amount of information has been useful.

3. **Chosen plaintext:** Eve gains temporary access to the encryption machine. She cannot open it to find the key; however, she can encrypt a

large number of suitably chosen plaintexts and try to use the resulting ciphertexts to deduce the key.

4. **Chosen ciphertext:** Eve obtains temporary access to the decryption machine, uses it to "decrypt" several strings of symbols, and tries to use the results to deduce the key.

A chosen plaintext attack could happen as follows. You want to identify an airplane as friend or foe. Send a random message to the plane, which encrypts the message automatically and sends it back. Only a friendly airplane is assumed to have the correct key. Compare the message from the plane with the correctly encrypted message. If they match, the plane is friendly. If not, it's the enemy. However, the enemy can send a large number of chosen messages to one of your planes and look at the resulting ciphertexts. If this allows them to deduce the key, the enemy can equip their planes so they can masquerade as friendly.

An example of a known plaintext attack reportedly happened in World War II in the Sahara Desert. An isolated German outpost every day sent an identical message saying that there was nothing new to report, but of course it was encrypted with the key being used that day. So each day the Allies had a plaintext-ciphertext pair that was extremely useful in determining the key. In fact, during the Sahara campaign, General Montgomery was carefully directed around the outpost so that the transmissions would not be stopped.

One of the most important assumptions in modern cryptography is **Kerckhoffs's Principle:** In assessing the security of a cryptosystem, one should always assume the enemy knows the method being used. This principle was enunciated by Auguste Kerckhoffs in 1883 in his classic treatise *La Cryptographie Militaire*. The enemy can obtain this information in many ways. For example, encryption/decryption machines can be captured and analyzed. Or people can defect or be captured. The security of the system should therefore be based on the key and not on the obscurity of algorithm used. Consequently, we always assume that Eve has knowledge of the algorithm that is used to perform encryption.

Symmetric and Public Key Algorithms

Encryption/decryption methods fall into two categories: **symmetric key** and **public key**. In symmetric key algorithms, the encryption and decryption keys are known to both Alice and Bob. For example, the encryption key is shared and the decryption key is easily calculated from it. In many cases, the encryption key and the decryption key are the same. All of the

classical (pre-1970) cryptosystems are symmetric, as are the more recent Data Encryption Standard (DES) and Rijndael (AES).

Public key algorithms were introduced in the 1970s and revolutionized cryptography. Suppose Alice wants to communicate securely with Bob, but they are hundreds of kilometers apart and have not agreed on a key to use. It seems almost impossible for them to do this without first getting together to agree on a key, or using a trusted courier to carry the key from one to the other. Certainly Alice cannot send a message over open channels to tell Bob the key, and then send the ciphertext encrypted with this key. The amazing fact is that this problem has a solution, called public key cryptography. The encryption key is made public, but it is computationally infeasible to find the decryption key without information known only to Bob. The most popular implementation is RSA (see Chapter 6), which is based on the difficulty of factoring large integers. Other versions (see Chapters 7 and 16) are due to ElGamal (based on the discrete log problem) and McEliece (based on error correcting codes).

Here is a nonmathematical way to do public key communication. Bob sends Alice a box and an unlocked padlock. Alice puts her message in the box, locks Bob's lock on it, and sends the box back to Bob. Of course, only Bob can open the box and read the message. The public key methods mentioned previously are mathematical realizations of this idea. Clearly there are questions of authentication that must be dealt with. For example, Eve could intercept the first transmission and substitute her own lock. If she then intercepts the locked box when Alice sends it back to Bob, Eve can unlock her lock and read Alice's message. This is a general problem that must be addressed with any such system.

Public key cryptography represents what is possibly the final step in an interesting historical progression. In the earliest years of cryptography, security depended on keeping the encryption method secret. Later, the method was assumed known, and the security depended on keeping the (symmetric) key private or unknown to adversaries. In public key cryptography, the method and the encryption key are made public, and everyone knows what must be done to find the decryption key. The security rests on the fact (or hope) that this is computationally infeasible. It's rather paradoxical that an increase in the power of cryptographic algorithms over the years has corresponded to an increase in the amount of information given to an adversary about such algorithms.

Public key methods are very powerful, and it might seem that they make the use of symmetric key cryptography obsolete. However, this added flexibility is not free and comes at a computational cost. The amount of computation needed in public key algorithms is typically several orders of magnitude more than the amount of computation needed in algorithms such as DES or Rijndael. The rule of thumb is that public key should not be used

for encrypting large quantities of data. For this reason, public key is used in applications where only small amounts of data must be processed (for example, digital signatures and sending keys to be used in symmetric key algorithms).

Within symmetric key cryptography, there are two types of ciphers: stream ciphers and block ciphers. In stream ciphers, the data are fed into the algorithm in small pieces (bits or characters), and the output is produced in corresponding small pieces. In block ciphers, however, a block of input bits is collected and fed into the algorithm all at once, and the output is a block of bits. In Section 2.11 we discuss an example of a stream cipher, namely linear feedback shift registers. Mostly we shall be concerned with block ciphers. In particular, we cover two very significant examples. The first is DES, and the second is Rijndael, which was selected in the year 2000 by the National Institute for Standards and Technology as the replacement for DES. Public key methods such as RSA can also be regarded as block ciphers.

Finally, we mention a historical distinction between different types of encryption, namely **codes** and **ciphers**. In a code, words or certain letter combinations are replaced by codewords (which may be strings of symbols). For example, the British navy in World War I used 03680C, 36276C, and 50302C to represent *shipped at, shipped by,* and *shipped from,* respectively. Codes have the disadvantage that unanticipated words cannot be used. A cipher, on the other hand, does not worry about the linguistic structure of the message but rather encrypts every string of characters, meaningful or not, by some algorithm. A cipher is therefore more versatile than a code. In the early days of cryptography, codes were commonly used, sometimes in conjunction with ciphers. They are still used today; covert operations are often given code names. However, any secret that is to remain secure needs to be encrypted with a cipher. In this book, we'll deal exclusively with ciphers.

Key Length

The security of cryptographic algorithms is a difficult property to measure. Most algorithms employ keys, and the security of the algorithm is related to how difficult it is for an adversary to determine the key. The most obvious approach is to try every possible key and see which ones yield meaningful decryptions. Such a type of attack is called a **brute force attack**. In a brute force attack, the length of the key is directly related to how long it will take to search the entire keyspace. For example, if a key is 16 bits long, then there are $2^{16} = 65536$ possible keys. The DES algorithm has a 56-bit key and thus has $2^{56} \approx 7.2 \times 10^{16}$ possible keys.

In many situations we'll encounter in this book, it will seem that a system can be broken by simply trying all possible keys. However, this is often easier said than done. Suppose you need to try 10^{30} possibilities and you have a computer that can do 10^9 such calculations each second. There are around 3×10^7 seconds in a year, so it would take a little more than 3×10^{13} years to complete the task, longer than the predicted life of the universe.

Longer keys are advantageous but are not guaranteed to make an adversary's task difficult. The algorithm itself also plays a critical role. Some algorithms might be able to be attacked by means other than brute force, and some algorithms just don't make very efficient use of their keys' bits. This is a very important point to keep in mind. Not all 128-bit algorithms are created equal!

For example, one of the easiest cryptosystems to break is the substitution cipher, which we discuss in Section 2.4. The number of possible keys is $26! \approx 4 \times 10^{26}$. In contrast, DES (see Chapter 4) has only $2^{56} \approx 7.2 \times 10^{16}$ keys. But it typically takes over a day on a specially designed computer to find a DES key. The difference is that an attack on a substitution cipher uses the underlying structure of the language, while the attack on DES is by brute force, trying all possible keys.

A brute force attack should be the last resort. A cryptanalyst always hopes to find an attack that is faster. Examples we'll meet are frequency analysis (for the substitution and Vigenère ciphers) and birthday attacks (for discrete logs).

We also warn the reader that just because an algorithm seems secure now, doesn't mean it will remain so. Human ingenuity has led to creative attacks on cryptographic protocols. There are many examples in modern cryptography where an algorithm or protocol was successfully attacked because of a loophole presented by poor implementation, or just because of advances in technology. The DES algorithm, which withstood 20 years of cryptographic scrutiny, ultimately succumbed to attacks by a well-designed parallel computer. Even as you read this book, research in quantum computing is underway, which could dramatically alter the terrain of future cryptographic algorithms.

For example, the security of several systems we'll study depends on the difficulty of factoring large integers, say of around 200 digits. Suppose you want to factor a number n of this size. The method used in elementary school is to divide n by all of the primes up to the square root of n. There are approximately 4×10^{97} primes less than 10^{100}. Trying each one is impossible. The number of electrons in the universe is estimated to be less than 10^{90}. Long before you finish your calculation, you'll get a call from the electric company asking you to stop. Clearly, more sophisticated factoring algorithms must be used, rather than this brute force type of attack. When RSA was invented, there were some good factoring algorithms available, but