Springer Undergraduate Mathematics Series

SUMS

Simon Rubinstein-Salzedo

Cryptography

Springer

Simon Rubinstein-Salzedo

Cryptography

Springer

Simon Rubinstein-Salzedo
Palo Alto, CA, USA

ISSN 1615-2085 ISSN 2197-4144 (electronic)
Springer Undergraduate Mathematics Series
ISBN 978-3-319-94817-1 ISBN 978-3-319-94818-8 (eBook)
https://doi.org/10.1007/978-3-319-94818-8

Library of Congress Control Number: 2018946572

Mathematics Subject Classification (2010): 11T71

Contents

1 A Quick Overview 1
- 1.1 The Setup of Cryptography 1
- 1.2 A Note on Sage 2
- 1.3 Problems 3

2 Cæsar Ciphers 5
- 2.1 What is a Cæsar Cipher? 5
- 2.2 Cracking the Cæsar Cipher 5
- 2.3 Some Terminology 6
- 2.4 Problems 7

3 Substitution Ciphers 9
- 3.1 What is a Substitution Cipher? 9
- 3.2 A Cheap Attack Against the Substitution Cipher 10
- 3.3 Frequency Analysis 12
- 3.4 Problems 19

4 A First Look at Number Theory 25
- 4.1 Prime Numbers and Factorization 25
- 4.2 Modular Arithmetic 29
- 4.3 The GCD and the Euclidean Algorithm 32
- 4.4 The Chinese Remainder Theorem 34
- 4.5 Multiplication Modulo m and the Totient Function 36
- 4.6 Problems 37

5 The Vigenère Cipher 41
- 5.1 The Vigenère Cipher 41
- 5.2 Algebraic Description of the Vigenère Cipher 42
- 5.3 Cracking the Vigenère Cipher: The Kasiski Examination 43
- 5.4 Distinguishing Substitution and Vigenère Ciphers 48

5.5 Friedman's Coincidence Index 49
5.6 Problems 52

6 The Hill Cipher 55
6.1 Matrices 55
6.2 Encrypting with Matrices 58
6.3 Attacking the Hill Cipher 60
6.4 Problems 62

7 Other Types of Ciphers 63
7.1 The Autokey Cipher 63
7.2 Transposition Ciphers 65
7.3 One-Time Pads 68
7.4 Introduction to Modern Cryptosystems 69
7.5 Problems 71

8 Big *O* Notation and Algorithm Efficiency 75
8.1 Big *O* Notation 75
8.2 Logarithms 76
8.3 Running Times of Algorithms 77
8.4 Problems 82

9 Abstract Algebra 85
9.1 Some Group Theory 85
9.2 Fields 86
9.3 Problems 87

10 A Second Look at Number Theory 89
10.1 Fermat's Little Theorem 89
10.2 Euler's Theorem 90
10.3 Primitive Roots and the Structure of $\mathbb{F}_p^\times$ 90
10.4 Numbers in Other Bases 92
10.5 Fast Computation of Powers in $\mathbb{Z}/m\mathbb{Z}$ 93
10.6 Multiplicative Functions 94
10.7 Problems 97

11 The Diffie–Hellman Key Exchange and the Discrete Logarithm Problem 99
11.1 The Diffie–Hellman Key Exchange 99
11.2 The Discrete Logarithm Problem 100
11.3 ElGamal Encryption 101
11.4 Symmetric Versus Asymmetric Key Cryptography 103
11.5 Shanks's Baby-Step Giant-Step Algorithm 103
11.6 Pohlig–Hellman 105
11.7 The Index Calculus 107
11.8 Problems 111

12 The RSA Cryptosystem 113
12.1 RSA 113
12.2 Attacking Naïve RSA 114
12.3 Padding 116
12.4 Blocking 116
12.5 Why Isn't Cryptography Solved? 118
12.6 Digital Signatures 118
12.7 Attacks on RSA 121
12.8 Security 122
12.9 Problems 124

13 Clever Factorization Algorithms and Primality Testing 127
13.1 Introduction to Clever Factorization Algorithms and Primality Tests 127
13.2 The Birthday Problem 128
13.3 Pollard ρ Algorithm 128
13.4 Smooth Numbers 130
13.5 The Quadratic Sieve 131
13.6 The Miller–Rabin Primality Test 135
13.7 The Agrawal–Kayal–Saxena Primality Test 137
13.8 Problems 139

14 Elliptic Curves 141
14.1 Rational Points on a Circle 141
14.2 Elliptic Curves 143
14.3 Elliptic Curves Over Finite Fields 146
14.4 Point Counting on Elliptic Curves 148
14.5 A Smattering of Group Theory 148
14.6 The Group Structure on $E(\mathbb{F}_p)$ 150
14.7 Elliptic Curve Diffie–Hellman 150
14.8 Why Elliptic Curves? 153
14.9 Problems 154

15 The Versatility of Elliptic Curves 157
15.1 Pollard's $p-1$ Factorization 157
15.2 Elliptic Curves Over $\mathbb{Z}/n\mathbb{Z}$ 160
15.3 Lenstra's Elliptic Curve Factorization 161
15.4 Elliptic Curves Over $\mathbb{Q}$ 164
15.5 Elliptic Curves and Diophantine Equations 165
15.6 Elliptic Curves Over $\mathbb{Z}$ 168
15.7 Problems 169

16 Zero-Knowledge Proofs 173
16.1 Introduction to Zero-Knowledge Proofs 173
16.2 What is a Zero-Knowledge Proof? 174

16.3 Ali Baba's Cave 176
16.4 Three-Colorability 177
16.5 Sudoku 180
16.6 Quadratic Residuosity 182
16.7 Discrete Logarithms and the Schnorr Protocol 183
16.8 Time Machines and Prover Tricks 185
16.9 Commitment Schemes 186
16.10 Problems 187

17 Secret Sharing, Visual Cryptography, and Voting 191
17.1 Secret Sharing 191
17.2 Shamir's Secret-Sharing Scheme 192
17.3 Blakley's Secret-Sharing Scheme 193
17.4 Visual Cryptography 194
17.5 Cryptographic Voting 196
17.6 An Outline of a Cryptographic Voting Protocol 197
17.7 Key Generation 198
17.8 Assigning Tokens 199
17.9 Encrypting and Verifying the Votes 199
17.10 The Decryption Mix-Net 201
17.11 Problems 203

18 Quantum Computing and Quantum Cryptography 207
18.1 Quantum Versus Classical Computers 207
18.2 Modifying Quantum States 209
18.3 A Quantum Cryptographic Protocol 210
18.4 Quantum Computers Defeat Classical Cryptography 212
18.5 The Quantum Fourier Transform 214
18.6 Continued Fractions 215
18.7 Back to Shor's Algorithm 217
18.8 Problems 219

19 Markov Chains 221
19.1 Introduction to Probability 221
19.2 Markov Chains 223
19.3 Stationarity 226
19.4 Letters and Markov Chains 227
19.5 Problems 229

20 Some Coding Theory 231
20.1 Introduction to Coding Theory 231
20.2 Linear Codes 233
20.3 Lexicographic Codes 234
20.4 Prisoners and Their Hats 238
20.5 Nim 241

20.6 Hamming Codes 242
20.7 Back to the Hats 243
20.8 Linearity 244
20.9 The Golay Codes 246
20.10 The Ternary Golay Code 246
20.11 The Binary Golay Code 248
20.12 Finite Simple Groups 249
20.13 Problems 249

References 251

Index 257

Introduction

This book stemmed from two classes: a 3-week class on cryptography I taught at Stanford's Education Program for Gifted Youth (EPGY) (now renamed "Stanford Pre-Collegiate Studies") in the summer of 2013, and a 10-week class on cryptography I taught at Euler Circle in the winter of 2016. My students in these classes were 14–17 years old, and as a result this book requires little mathematical background. My students worked so hard, and we got quite far and covered many interesting topics. I was pleased with how much they learned in such a short period of time. My original notes, from the EPGY class, only covered what I was able to get through in twelve days of lectures, and then I expanded them to include the additional material we covered in the Euler Circle class. I have expanded the notes further to include several other topics that I find interesting and important.

While this book is officially about cryptography, it is really an excuse for me to share some fascinating mathematics that I hope readers will enjoy. It isn't intended to be a comprehensive or encyclopedic treatise on the subject, and many important topics in modern practice have been omitted. Topics are selected for being mathematically exciting, and that sometimes meant skipping other topics of greater practical significance, and also covering some topics as "proof of concept" rather than giving the most secure or up-to-date versions of things. I encourage readers who want to see how everything is done in practice today to follow up by reading the latest papers and developments; since this is a fast-moving and active area of research, I expect that several things I have written will already be obsolete by the time the book goes to press.

As in any mathematics book, the problems are very important. They are intended to be doable but challenging, and ideally several people will work on the problems together and share ideas.

Originally, cryptography and cryptanalysis were thought to be linguistic topics, and the best codebreakers were people who knew a lot about language. While that was true for breaking fairly easy ciphers, it became less and less true as cryptography advanced. In the twentieth century, mathematicians became the leaders in cryptanalysis. For example, Polish and British mathematicians before and during World War II were spectacularly successful in breaking the German Enigma code using some then-recent advances in mathematics, while using essentially no linguistic inputs. (Indeed, it was not even necessary to understand German in order to decode the German messages.)

Starting in the 1970s, mathematicians also took over the codemaking industry with the discoveries of Diffie and Hellman, and then Rivest, Shamir, and Adleman. They used ideas from number theory to develop exciting new cryptosystems that are still in widespread use today. We shall study them in detail, introducing all the relevant background along the way. In addition, we shall look at methods of attacking them.

In this book, we shall study cryptography from its earliest roots 2000 years ago to the main cryptosystems still used today for secure online communication. Along the way, we will encounter a lot of truly delightful mathematics. Most of it is number theory, but topics in combinatorics, probability, abstract algebra, and others will also make cameo appearances. I think that cryptography is a great place to learn about these subjects, as they are developed in order to tackle specific problems that arise naturally.

I hope you enjoy reading this book as much as I enjoyed writing it and teaching the classes!

I would like to thank Porter Adams, Alon Amit, Sohini Kar, Yelena Mandelshtam, Nitya Mani, Smiti Mittal, Ken Wanlass, Jonathan Webster, and the students from my classes for reading earlier versions of this book carefully, making suggestions and corrections that improved this book. I would also like to thank Eva Goedhart, Ho Chung Siu, and Jonathan Sorenson for teaching classes out of earlier versions of the book and giving me feedback.

From time to time, I have provided hints for the exercises. These hints are in Rot13, which is a Cæsar shift of 13. If you want to see the hints, you can either perform the Rot13 yourself, or you can go to http://www.rot13.com/ to have it done automatically.

Chapter 1
A Quick Overview

1.1 The Setup of Cryptography

Cryptography is the study of methods of secure communication between two parties. Typically, there are two parties who want to send messages to each other, but they want to avoid the possibility of a third party understanding these messages should they fall into the wrong hands.

It is traditional in cryptography to name our protagonists and antagonist, and they have standard names that we will be referring to throughout the book. Our protagonists are named Alice and Bob. They are simple people: they want nothing more out of life than to be able to send secure messages to each other. However, as we shall see, their task is not so easy, and it is necessary for them to learn some fascinating mathematics in order to accomplish their task.

On the other hand, we have Eve, our antagonist. Eve desperately wants to read and understand Alice and Bob's correspondence. She too will have to learn some fascinating mathematics in order to keep up with Alice and Bob's ever more challenging codes.

Cryptography is the battle between Alice and Bob on one side, and Eve on the other. At various times in history, Alice and Bob have had the upper hand. At other times, Eve has been on top. At present, it seems that Alice and Bob are winning, but Eve is hard at work trying to regain her lead. Who knows what tricks the two sides will come up with in the future?

Our standing assumption will be that any time a person sends a message, that person has to send it over a public medium, so that anyone who wishes can pick it up. So, Eve can receive any message that Alice and Bob send to each other. The point, then, is to make it so that even though Eve can see the message, it just looks like gibberish to her: she can't access the *content* of the message.

So, how can Alice and Bob do that? Suppose Alice wishes to send the message `HELLO` to Bob. She can't just send `HELLO`, or else Eve will be able to read it. So, she has to send something else, so that Bob knows it means `HELLO`, and Eve does not.

In order for them to do that, Alice and Bob have to agree on a *protocol* beforehand: Alice will modify some or all of the letters in a prespecified manner, and Bob will know how to recover the original text. Eve, not having attended their protocol meeting, will not know how the text was modified and will have to figure it out by (clever) guessing.

The first code we'll look at, in the next chapter, will be the Cæsar cipher, Julius Cæsar's preferred method of cryptography.

1.2 A Note on Sage

Throughout this book, we will see various computer programs that help us do calculations that would be extremely tedious or impossible to perform by hand. Most of these are written in Sage. Sage is a free computer algebra system that can be downloaded from http://www.sagemath.org/, or used online at https://www.cocalc.com/ with a free account.

Sage is an easy-to-use programming language that also functions as a very advanced calculator. One can ask it to perform simple arithmetical operations: if I type

```
5*6
```

then it outputs 30, as expected. It also has much more complicated commands built in. For example, we can ask it to factor numbers:

```
factor(2^32+1)
```

and it outputs

```
641 * 6700417
```

which is indeed the prime factorization of $2^{32} + 1$. Sage also has the standard features of a programming language: `if` statements, `for` and `while` loops, and so forth. The Sage snippets displayed throughout the book ought to be easy to follow and modify for your own purposes, and there are various tutorials available online, such as [Tea17]. Sage is built on the Python programming language, so it follows Python syntax, and standard Python commands can also be used in Sage.

1.3 Problems

(1) Generally, we use codes in order to prevent unwanted third parties from reading our messages. Are there any other reasons why one might choose to write in code? Come up with as many as you can.

(2) Suppose Alice wishes to send a message to Bob, in such a way that Bob can be sure that Eve has not intercepted or tampered with the message. What are some low-tech mechanisms she can do so that Bob can certify that only Alice could have sent the message, and Eve hasn't read or modified it?

Chapter 2
Cæsar Ciphers

2.1 What is a Cæsar Cipher?

Let's say that Alice wants to send Bob the message

`cryptography is fun.`

What she can do is to replace every letter by a different letter. Perhaps, she thinks up the following simple scheme that Julius Cæsar is said to have used: she replaces every letter by the one that comes three places later in the alphabet, so every `a` becomes a `D`, every `b` becomes an `E`, and so forth. Eventually, we reach the end of the alphabet, so we have to wrap around, so every `w` becomes a `Z`, but then every `x` becomes an `A`, every `y` becomes a `B`, and every `z` becomes a `C`. Under this scheme, Alice's message becomes

`FUBSWRJUDSKB LV IXQ.`

Note that we have used lowercase letters for the original message, and uppercase letters for the encoded one. We will do this throughout the book.

Now, Alice sends the encoded message instead of the original one. If Bob doesn't know what Alice has done, he will be quite confused, because the message that she sent doesn't make any sense. And Eve will be similarly stumped. However, if Bob knows that Alice was going to apply this scheme to produce the message, he can figure out what it is supposed to say, for he can replace each letter by the one that comes three places *earlier* in the alphabet, so every `A` is actually an `x`, every `B` is a `y`, every `C` is a `z`, every `D` is an `a`, and so forth.

We call this code the Cæsar cipher.

2.2 Cracking the Cæsar Cipher

Now, is this a good way to send codes? It was state-of-the-art cryptography in Julius Cæsar's day, but by modern standards, it's a terribly weak system. The problem is that if Eve decides she wants to break it, and she suspects that Alice and Bob are using a Cæsar cipher, it won't be too hard for her to do so. She might not know exactly which Cæsar cipher they are using: she might not know that Alice shifted every letter over by 3, but she suspects that every letter is shifted over by *the same amount* n, so that Alice is replacing every letter with the letter that comes n letters further along in the alphabet (for some number n). There aren't that many possibilities for her to try. We will now expand our definition of a Cæsar cipher: instead of requiring it to be a shift by exactly three letters, we will allow a shift by *any* number of letters.

Exercise How many possible Cæsar ciphers are there?

Now, Eve can't just blindly try all the possibilities; she also needs some method of determining whether she has it right. How can Eve tell when she has got it right? That isn't too hard: it's right when she recovers a message that makes sense.

Let us suppose that she received the encoded message

```
FUBSWRJUDSKB LV IXQ
```

and guessed (wrongly) that it was the Cæsar cipher obtained by shifting each letter by six instead of by three. Then, she would end up with the message

```
zovmqldoxmev fp crk.
```

She should suspect that this is wrong, because those don't look like English words (or words in any other language she is familiar with). Most of the time, only one choice of shift will result in something that looks like English words, and that will be the correct one. So, in other words, she can just keep on trying until she gets some words back.

Exercise See if you can find a message in English that can be encoded with a Cæsar cipher to look like another English message.

You will probably find the previous exercise to be very difficult. It is very rare for a Cæsar cipher to be ambiguous. (However, there are some examples, such as `sleep` and `bunny`, which are Cæsar shifts of one another.) As a result, it is unlikely that Eve will correctly guess that a message has been sent using a Cæsar cipher, but still decipher it incorrectly. So, in conclusion, it's easy for Eve to break the Cæsar cipher (and know that she has broken them), and Alice and Bob should look for better ways of encoding messages.

2.3 Some Terminology

Now that we have seen the basics of cryptography, it is time to introduce some terminology. As we have seen, Alice and Bob want to send each other messages, but they end up sending different messages from the ones they actually wanted to send. We will call the message they actually want to send the *plaintext*, and the message they actually send the *ciphertext*. A *cryptosystem* is a pair of algorithms, one to convert from a plaintext to a ciphertext, and the other to convert from a ciphertext back to a plaintext. In order to generate the ciphertext from the plaintext, two ingredients are needed: the protocol for encoding the message, and a specific *key*. In the case of the Cæsar cipher, the Cæsar cipher is the protocol, and the key determines the size of the shift.

It is important to distinguish between the encoding protocol and the key. A good code should satisfy *Kerckhoff's principle*, which says that a good cryptosystem should remain secure if the attacker (in this case Eve) knows the protocol and everything about the method of encryption, with the exception of the key. Kerckhoff's principle is in contrast to *security through obscurity*, in which it is assumed that the attacker is unable to determine the protocol used for generating the ciphertext. Frequently, peculiar features of the ciphertext will allow the attacker to make a good guess about the protocol used. We shall return to this topic later, once we have studied several more cryptosystems.

In order to distinguish between the plaintext and the ciphertext, it is common to use uppercase letters for the ciphertext and lowercase letters for the plaintext. Sometimes, we will write the plaintext and ciphertext on two lines, as in the following:

```
 Plaintext: cryptography is fun
Ciphertext  ETARVQITCRJA KU HWP
```

When typesetting, using a monospaced font is recommended, so that the letters line up nicely.

It is also worth distinguishing between *cryptography* and *cryptanalysis*. *Cryptography* is the writing of hidden messages, and *cryptanalysis* is the analysis of hidden messages. Thus, Alice and Bob engage in cryptography, while Eve engages in cryptanalysis. However, it is important for both sides to be well-versed in both subjects. Alice and Bob need to know the state-of-the-art attacks on their cryptosystem, so that when a really dangerous attack comes along, they know to adopt a different protocol. On the other hand, Eve needs to be booked up on the latest news in cryptography, so that she can make a good guess about the protocol Alice and Bob are using.

2.4 Problems

(1) Encrypt the message

```
     For duty, duty must be done;
    the rule applies to everyone.
```

using a Cæsar cipher with a shift of six letters. (Don't worry about encrypting the punctuation.)

(2) The message

```
ESTYRD LCP DPWOZX LD ESPJ DPPX.
```

is a ciphertext encrypted using a Cæsar cipher (the shift amount is not provided). What is the plaintext message?

(3) Can you find two English words *other than* `sleep` and `bunny`, with at least four letters, that encrypt to each other under a Cæsar cipher? What are the longest such words you can find?

(4) Alice sends Bob a ciphertext encrypted using a Cæsar cipher. Bob knows the shift amount, but being a beginner in cryptography at this stage, he accidentally *encrypts* it rather than decrypting it. Fortunately, by doing so, he still recovers the original plaintext. What was the shift amount?

(5) Actually, I tricked you in problem 4. The message was actually in Hebrew, rather than in English. The Hebrew alphabet consists of 22 letters. What was the shift amount actually?

(6) Suppose Alice starts with a plaintext message and encrypts it using a Cæsar cipher with a shift of k letters. Since she is also just learning about cryptography at the moment, she explores what happens when she encrypts the encrypted ciphertext, using the same shift of k. She does this repeatedly, and eventually she finds that she has decrypted the message. In terms of k, how many times did Alice have to encrypt the message before it became decrypted? (You may assume that the message is in English this time, so that the alphabet has 26 letters.)

Chapter 3
Substitution Ciphers

3.1 What is a Substitution Cipher?

As we saw in the last chapter, the Cæsar cipher isn't very good. (This isn't too surprising: we should expect that cryptography has seen some major advances since its beginnings around 2000 years ago.) But it is easy to come up with an improvement: instead of shifting everything over by a fixed amount, we can have every letter stand for a different letter, at random perhaps. So, for example, we could consider the following set of substitutions:

```
abcde fghij klmno pqrst uvwxy z
XHZRW FGPTY JOCAE ULNKQ IVSMD B
```

The first line above denotes a plaintext character, and the second line denotes the corresponding ciphertext character. This means that, for example, every time we want to write an `a`, we actually write an `X`, and every time we actually want to write a `b`, we write an `H`, and so on. Hence, our plaintext message

```
cryptography is fun
```

becomes

```
ZNDUQEGNXUPD TK FIA
```

in the ciphertext.

If Alice sends this message, then Bob can decrypt it by looking up in the table. The first letter he sees is a `Z`, so he looks for a `Z` in the bottom row of the table and finds it sitting under the `c`, so the `Z` should have been a `c`. He keeps doing this for every letter until he has decrypted the whole message. He may find the task easier if he sorts the table by the ciphertext character rather than the plaintext character:

```
ABCDE FGHIJ KLMNO PQRST UVWXY Z
nzmyo fgbuk sqxrl htdwi pveaj c
```

In contrast to Bob, Eve has difficulty in deciphering the message since she doesn't know which letter substitutions were used.

This generalization of the Cæsar cipher is called the substitution cipher. It is a lot better than the Cæsar cipher largely because there are a lot more possible substitution ciphers than there are Cæsar ciphers. In fact, there are 26! (26 factorial) of them, as opposed to just 26. Note that $26! = 403291461126605635584000000$, or around 4×10^{26}. Eve won't want to try all of them, even with a computer. At the very least, it will require some cleverness in order to break a substitution cipher. However, as we shall see, the number of possible keys is not the only determining factor in the difficulty of breaking a ciphertext.

Also, note that in a substitution cipher, some letters might get encoded as themselves. (In the above example, this happens with the letters `f`, `g`, and `v`.) Is this a flaw in the strength of the substitution cipher?

The answer is a most emphatic **no**. Indeed, allowing letters to code for themselves is a *strength* of the substitution cipher. This may be counterintuitive, so let us pause for a moment to think about why it is actually a good idea. The reason is that it doesn't give away any clues. If a letter were not allowed to code for itself, then Eve may be able to use that as a clue. Perhaps she is debating whether `I` stands for `i` or `u` (the latter being correct in the above substitution cipher). Then, she would already know the answer, since the `I` wouldn't be allowed to code for itself. But as it is, we *are* allowing letters to code for themselves, so Eve doesn't get any hints like that for free.

3.2 A Cheap Attack Against the Substitution Cipher

When we think about cryptography, we're definitely rooting for Alice and Bob. After all, they're the good guys. However, in order to ensure their success, it is very important that we sometimes stop to think about things from Eve's point of view. She doesn't have anything else to do besides trying to crack codes, so she can devote a lot of time to thinking up clever ways to attack any codes or ciphers that Alice and Bob come up with. So, she isn't just about to give up because the problem initially looks hard.

So far, we have given her a huge hint: some of the words in our message are very short. There aren't any 1-letter words in our previous message, but suppose for a moment that there were some, perhaps because we wish to send the new message

```
i have a truly marvelous demonstration
           of this proposition
```

After we encode it, we obtain the ciphertext

```
T PXVW X QNIOD CXNVWOEIK RWCEAKQNXQTEA
           EF QPTK UNEUEKTQTEA
```

How can Eve try to make progress on cracking this code? Well, we have two 1-letter words in there: the first and third words. There are only two 1-letter words in English: `a` and `I`. So, assuming that Alice and Bob are speaking proper English, `T` and `X` must be `a` and `i`, in some order. Originally, Eve had 26 choices for `T` and then 25 left for `X`, but now she is down from 26×25 to 2: a huge improvement!

Let's see how easily Eve can crack this substitution cipher: suppose she guesses correctly that `T` means `i` and that `X` means `a`. Once she makes all these substitutions, she is left with the following (we'll write the ones she has figured out in lowercase letters and keep the ones she's still working on in uppercase):

```
i PaVW a QNIOD CaNVWOEIK RWCEAKQNaQiEA
          EF QPiK UNEUEKiQiEA
```

Well, that takes care of the 1-letter words. It helps some, but there's a lot more work to do. Now how about the 2-letter word? It doesn't contain an `a` or an `i`, so that cuts down on the possibilities. What is a good candidate for what it might be? There are several reasonable possibilities: perhaps `of` or `or` or `be` would be good choices. So maybe she checks all of them to see which ones resemble English. Since we're trying to do this relatively quickly by hand (without the help of a computer), let's say she gets it right and guesses that `EF` is supposed to be `of`. Then, making those substitutions, we have

```
i PaVW a QNIOD CaNVWOoIK RWCoAKQNaQioA
          of QPiK UNoUoKiQioA
```

What next? Now, she might notice that there are two words that end with `QioA`. What's a common ending for words of this form? The logical choices are `sion` and `tion`. Both end with `n`, so let's see what happens when we replace `A` with `n`. We get

```
i PaVW a QNIOD CaNVWOoIK RWConKQNaQion
          of QPiK UNoUoKiQion
```

That wasn't a huge help, but let's see what happens if we try replacing `Q` with `s` and `t`. With `s` we get

```
i PaVW a sNIOD CaNVWOoIK RWConKsNasion
          of sPiK UNoUoKision
```

Now, we can see if there are any English words with these letter combinations. My mother's crossword puzzle solver tells me that there are no words of the form `??o?o?ision` so this was wrong. (Of course, Eve also has access to a crossword puzzle solver, so she can do that too.) Let's try with `t` instead. We get

```
i PaVW a tNIOD CaNVWOoIK RWConKtNation
          of tPiK UNoUoKition
```

Checking again, we find that there is exactly one word of the form `??o?o?ition`, namely `proposition`. Assuming this is right, we get to fill in lots of stuff now: we get

```
i PaVW a trIOD CarVWOoIs RWConstration
          of tPis proposition
```

Again, we check which words can fit: there are two possible words of the form `t?is`, namely, `this` and `tris`. Of course, `this` is much more likely, so that's what we should guess, but actually we already know that `tris` has to be wrong, since we've already found the `r`. So `P` must mean `h`. Similarly, `RWConstration` must be `demonstration`. Once we have substituted all that, we get

```
i haVe a trIOD marVeOoIs demonstration
          of this proposition
```

Continuing on in the same manner, we find that `marVeOoIs` is `marvelous`, giving us

```
i have a trulD marvelous demonstration
          of this proposition
```

Now it's not hard to guess that `D` should be `y`, giving us our final message of

```
i have a truly marvelous demonstration
          of this proposition
```

So we see that with a bit of clever guessing, Eve can crack this substitution cipher, at least some of the time. However, observe that Alice and Bob were being very generous to Eve by telling her where the spaces were, and thus how long the words are. So, they can be smarter about it, in one of two ways: they could smash everything together and count on each other to be able to reconstruct the spaces, or they could encode the spaces as well. Either of these methods will make life substantially harder for Eve. The standard method is simply to remove the spaces altogether; indeed, a bit of thought reveals that this is much more challenging for Eve to attack.

Exercise If we were to encode spaces as a "27th letter," how could Eve modify the method above in order to break a substitution cipher?

3.3 Frequency Analysis

Still, Eve still has some tricks up her sleeve, even if Alice and Bob delete the spaces before encoding the message. The main tool she can employ is frequency analysis. The idea is that the letters in English don't occur equally often: `e` is the most common letter, and `z` is the least common. See Table 3.1 for a table of the frequency of the letters in English, according to Wikipedia.

Eve can use her knowledge of letter frequencies to her advantage by assuming that the most common letter is an `e`, the second most common letter is a `t`, and so forth. Now, this might not be exactly right, since in a relatively small body of text, the proportions might be off by a bit. But it's still a good place to start. The longer the message, the easier time Eve will have of decoding it, since in longer messages, the proportions should be closer to what they are for English. Let's try out an example. Suppose Alice sends the message

Table 3.1 Table of letter frequencies in English

Letter	Frequency (%)	Rank
E	12.702	1
T	9.056	2
A	8.167	3
O	7.507	4
I	6.966	5
N	6.749	6
S	6.327	7
H	6.094	8
R	5.987	9
D	4.253	10
L	4.025	11
C	2.782	12
U	2.758	13
M	2.406	14
W	2.360	15
F	2.228	16
G	2.015	17
Y	1.974	18
P	1.929	19
B	1.492	20
V	0.978	21
K	0.772	22
J	0.153	23
X	0.150	24
Q	0.095	25
Z	0.074	26

```
PCFHN RJSDJ IXKXP YDCHI DLJDY HYNTJ IGFBD XIFVS
HRSDH DLXRH ITMJB JLXJZ FMIJO HHSDJ VXJTT DLXIX
GJIVJ ATXYX HYTXP SLPBD HIFIJ DDTXD LXGHC CDHJY
HYNTJ IDNSX DLXYT NKVHC THIMS XTBHS HSAHJ IMHCD
LXZPK DHIFO XSPNB HCAPB GJIKV MXZPB PSOJY TJSDL
XLNGH NIHCC PXTMP SORLP KLBHN SMBKH SDIJM PKDHI
FKHHT SXBBH CYJOX DJAHN DDHDI XYJSD LXBKP XSKXH
CQNTT PXSDL XXGPS XSDGN BPKHR PDHCG JKJNT JFRLH
RIHDX HCWNX XSJSS XDLXY JDLHB HCYJM MFJBI XSMXI
XMAFA HNKPK JNTDB DFTXH CDLXA PBLHY HCBHM HIJSM
```

```
GJSDL XMJBL HCJMH IBJFM PZXBD XMHCW NJKVX IFSJI
IJDPZ XYHRX IBHCM PKVXS BJSMD LJKVX IJFZP KDHIX
GGJSN XTYXJ VLJNS DPSOY XZXIP TDLHG JBJWN PSJBJ
SMMHK DHIBJ KLXZX IXTTD NYYXI JSMDX SSFBH SMJSP
XTMXC HXJSD LHSFD IHTTH YXJSM GPBDX IONPU HDJLD
JVXHC DLXBX XTXGX SDBJT TDLJD PBCNB PATXG XTDDL
XGJTT MHRSP SJYPY VPSHI KINKP ATXBX DDLXG DHBPG
GXIJS MDJVX HCCDL XBKNG JSMJL XJZFM IJOHH SPBDL
XIXBP MNNG
```

We can count the number of instances of each letter in the ciphertext. We can do this by hand, but it's boring, so I wrote a Python program to do it for me:

```
alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
ciphertext =
"PCFHN RJSDJ IXKXP YDCHI DLJDY HYNTJ IGFBD
XIFVS HRSDH DLXRH ITMJB JLXJZ FMIJO HHSDJ
VXJTT DLXIX GJIVJ ATXYX HYTXP SLPBD HIFIJ
DDTXD LXGHC CDHJY HYNTJ IDNSX DLXYT NKVHC
THIMS XTBHS HSAHJ IMHCD LXZPK DHIFO XSPNB
HCAPB GJIKV MXZPB PSOJY TJSDL XLNGH NIHCC
PXTMP SORLP KLBHN SMBKH SDIJM PKDHI FKHHT
SXBBH CYJOX DJAHN DDHDI XYJSD LXBKP XSKXH
CQNTT PXSDL XXGPS XSDGN BPKHR PDHCG JKJNT
JFRLH RIHDX HCWNX XSJSS XDLXY JDLHB HCYJM
MFJBI XSMXI XMAFA HNKPK JNTDB DFTXH CDLXA
PBLHY HCBHM HIJSM GJSDL XMJBL HCJMH IBJFM
PZXBD XMHCW NJKVX IFSJI IJDPZ XYHRX IBHCM
PKVXS BJSMD LJKVX IJFZP KDHIX GGJSN XTYXJ
VLJNS DPSOY XZXIP TDLHG JBJWN PSJBJ SMMHK
DHIBJ KLXZX IXTTD NYYXI JSMDX SSFBH SMJSP
XTMXC HXJSD LHSFD IHTTH YXJSM GPBDX IONPU
HDJLD JVXHC DLXBX XTXGX SDBJT TDLJD PBCNB
PATXG XTDDL XGJTT MHRSP SJYPY VPSHI KINKP
ATXBX DDLXG DHBPG GXIJS MDJVX HCCDL XBKNG
JSMJL XJZFM IJOHH SPBDL XIXBP MNNG"
counter = [0]*26
for char in ciphertext:
    num = alphabet.find(char)
    counter[num] += 1
print counter
```

It provides me with the following frequency table:

Letter	Freq	Letter	Freq	Letter	Freq
A	9	J	66	S	50
B	37	K	23	T	34
C	24	L	34	U	1
D	62	M	31	V	12
E	0	N	27	W	3
F	17	O	8	X	82
G	21	P	39	Y	22
H	67	Q	1	Z	9
I	41	R	9		

Now, we could just try to make substitutions based on this table naïvely, expecting that `X` should be `e`, `H` should be `t`, `J` should be `a`, and so forth. However, doing this immediately is a little risky: we may make a mistake early on and then be chasing nonsense. We can improve by looking not just at the individual letters, but also at the two-letter and three-letter combinations. The most common two-letter combination (bigram) is `th`, and the most common three-letter combination (trigram) is `the`. The most common bigram in the ciphertext is `DL` at 22, followed by `LX` at 19; the most common trigram is `DLX` at 16, far above `JSM` at 7. These observations strongly suggest that `DLX` is `the`. This is not quite consistent with the single-letter frequency analysis, since `D` is only the fourth most common letter in the ciphertext. But it is convincing enough!

Once we make these substitutions, we obtain

```
PCFHN RJStJ IeKeP YtCHI thJtY HYNTJ IGFBt
eIFVS HRStH theRH ITMJB JheJZ FMIJO HHStJ
VeJTT theIe GJIVJ ATeYe HYTeP ShPBt HIFIJ
ttTet heGHC CtHJY HYNTJ ItNSe theYT NKVHC
THIMS eTBHS HSAHJ IMHCt heZPK tHIFO eSPNB
HCAPB GJIKV MeZPB PSOJY TJSth ehNGH NIHCC
PeTMP SORhP KhBHN SMBKH StIJM PKtHI FKHHT
SeBBH CYJOe tJAHN ttHtI eYJSt heBKP eSKeH
CQNTT PeSth eeGPS eStGN BPKHR PtHCG JKJNT
JFRhH RIHte HCWNe eSJSS etheY JthHB HCYJM
MFJBI eSMeI eMAFA HNKPK JNTtB tFTeH CtheA
PBhHY HCBHM HIJSM GJSth eMJBh HCJMH IBJFM
PZeBt eMHCW NJKVe IFSJI IJtPZ eYHRe IBHCM
PKVeS BJSMt hJKVe IJFZP KtHIe GGJSN eTYeJ
VhJNS tPSOY eZeIP TthHG JBJWN PSJBJ SMMHK
tHIBJ KheZe IeTTt NYYeI JSMte SSFBH SMJSP
eTMeC HeJSt hHSFt IHTTH YeJSM GPBte IONPU
HtJht JVeHC theBe eTeGe StBJT TthJt PBCNB
PATeG eTtth eGJTT MHRSP SJYPY VPSHI KINKP
ATeBe ttheG tHBPG GeIJS MtJVe HCCth eBKNG
JSMJh eJZFM IJOHH SPBth eIeBP MNNG
```

At this stage, we have to start making more guesses. The most popular letter we haven't worked out yet is `a`, and the most popular remaining letters in the ciphertext are `H` and `J`, so we might guess that one of them means `a`. We don't really know if this is correct, but we can experiment. (Eve has to be very patient if she is going to break this cipher.)

Which of these is more convincing? Well, `tha` and `tho` are both common letter combinations, but `that` is certainly more common than `thot`. So if we find instances of `thHt`, then `H` is probably `a`. If we instead find `thJt`, then `J` is probably `a`. Indeed, we find `thJt`, for instance in the first line, fifth block.

The second most common trigram in English is `and`. Indeed, the second most common trigram in the ciphertext is `JSM`, which begins with a `J`, which we believe to be `a`. So, let's try substituting those. Furthermore, let's replace the next most common letter `H` with `o`, the next most common English letter we have not yet dealt with. We then obtain

```
PCFoN Ranta IeKeP YtCoI thatY oYNTa IGFBt
eIFVn oRnto theRo ITdaB aheaZ FdIaO oonta
VeaTT theIe GaIVa ATeYe oYTeP nhPBt oIFIa
ttTet heGoC CtoaY oYNTa ItNne theYT NKVoC
ToIdn eTBon onAoa IdoCt heZPK toIFO enPNB
oCAPB GaIKV deZPB PnOaY Tanth ehNGo NIoCC
PeTdP nORhP KhBoN ndBKo ntIad PKtoI FKooT
neBBo CYaOe taAoN ttotI eYant heBKP enKeo
CQNTT Penth eeGPn entGN BPKoR PtoCG aKaNT
aFRho RIote oCWNe enann etheY athoB oCYad
dFaBI endeI edAFA oNKPK aNTtB tFTeo CtheA
PBhoY oCBod oIand Ganth edaBh oCado IBaFd
PZeBt edoCW NaKVe IFnaI IatPZ eYoRe IBoCd
PKVen Bandt haKVe IaFZP KtoIe GGanN eTYea
VhaNn tPnOY eZeIP TthoG aBaWN PnaBa nddoK
toIBa KheZe IeTTt NYYeI andte nnFBo ndanP
eTdeC oeant honFt IoTTo Yeand GPBte IONPU
otaht aVeoC theBe eTeGe ntBaT Tthat PBCNB
PATeG eTtth eGaTT doRnP naYPY VPnoI KINKP
ATeBe ttheG toBPG GeIan dtaVe oCCth eBKNG
andah eaZFd IaOoo nPBth eIeBP dNNG
```

How can she confirm that she is on the right track? She can look at the result so far and see if it looks like plausible English. We don't find many bizarre letter combinations (and those we do find probably have spaces between them anyway), and we find some standard ones like "ten" that we didn't force to be there. So, this suggests that she is probably doing well.

At this stage, it's probably worth abandoning frequency tables and start looking for potential words. We find `theIe` and `theBe` in various places, which gives us a clue that `I` and `B` represent `r` and `s`, in some order. One could look for more clues, or just try one at random and see if anything looks weird. Let us try to replace `I` with `r` and `B` with `s` (for no other reason than that's the first thing I thought of), we obtain

```
PCFoN Ranta reKeP YtCor thatY oYNTa rGFst
erFVn oRnto theRo rTdas aheaZ FdraO oonta
VeaTT there GarVa ATeYe oYTeP nhPst orFra
ttTet heGoC CtoaY oYNTa rtNne theYT NKVoC
Tordn eTson onAoa rdoCt heZPK torFO enPNs
oCAPs GarKV deZPs PnOaY Tanth ehNGo NroCC
PeTdP nORhP KhsoN ndsKo ntrad PKtor FKooT
nesso CYaOe taAoN ttotr eYant hesKP enKeo
CQNTT Penth eeGPn entGN sPKoR PtoCG aKaNT
aFRho Rrote oCWNe enann etheY athos oCYad
dFasr ender edAFA oNKPK aNTts tFTeo CtheA
PshoY oCsod orand Ganth edash oCado rsaFd
PZest edoCW NaKVe rFnar ratPZ eYoRe rsoCd
PKVen sandt haKVe raFZP Ktore GGanN eTYea
VhaNn tPnOY eZerP TthoG asaWN Pnasa nddoK
torsa KheZe reTTt NYYer andte nnFso ndanP
eTdeC oeant honFt roTTo Yeand GPste rONPU
otaht aVeoC these eTeGe ntsaT Tthat PsCNs
PATeG eTtth eGaTT doRnP naYPY VPnor KrNKP
ATese ttheG tosPG Geran dtaVe oCCth esKNG
andah eaZFd raOoo nPsth eresP dNNG
```

There are several other tipoffs we can find. Let's mark some of them in red:

```
PCFoN Ranta reKeP YtCor thatY oYNTa rGFst
erFVn oRnto theRo rTdas aheaZ FdraO oonta
VeaTT there GarVa ATeYe oYTeP nhPst orFra
ttTet heGoC CtoaY oYNTa rtNne theYT NKVoC
Tordn eTson onAoa rdoCt heZPK torFO enPNs
oCAPs GarKV deZPs PnOaY Tanth ehNGo NroCC
PeTdP nORhP KhsoN ndsKo ntrad PKtor FKooT
nesso CYaOe taAoN ttotr eYant hesKP enKeo
CQNTT Penth eeGPn entGN sPKoR PtoCG aKaNT
aFRho Rrote oCWNe enann etheY athos oCYad
dFasr ender edAFA oNKPK aNTts tFTeo CtheA
PshoY oCsod orand Ganth edash oCado rsaFd
PZest edoCW NaKVe rFnar ratPZ eYoRe rsoCd
PKVen sandt haKVe raFZP Ktore GGanN eTYea
VhaNn tPnOY eZerP TthoG asaWN Pnasa nddoK
torsa KheZe reTTt NYYer andte nnFso ndanP
eTdeC oeant honFt roTTo Yeand GPste rONPU
otaht aVeoC these eTeGe ntsaT Tthat PsCNs
PATeG eTtth eGaTT doRnP naYPY VPnor KrNKP
```

```
ATese ttheG tosPG Geran dtaVe oCCth esKNG
andah eaZFd raOoo nPsth eresP dNNG
```

Plausible guesses are that `V` is `k`, `P` is `i`, `C` is `f`, and `N` is `u`. Trying this, we obtain

```
ifFou Ranta reKei Ytfor thatY oYuTa rGFst
erFkn oRnto theRo rTdas aheaZ FdraO oonta
keaTT there Garka ATeYe oYTei nhist orFra
ttTet heGof ftoaY oYuTa rtune theYT uKkof
Tordn eTson onAoa rdoft heZiK torFO enius
ofAis GarKk deZis inOaY Tanth ehuGo uroff
ieTdi nORhi Khsou ndsKo ntrad iKtor FKooT
nesso fYaOe taAou ttotr eYant hesKi enKeo
fQuTT ienth eeGin entGu siKoR itofG aKauT
aFRho Rrote ofWue enann etheY athos ofYad
dFasr ender edAFA ouKiK auTts tFTeo ftheA
ishoY ofsod orand Ganth edash ofado rsaFd
iZest edofW uaKke rFnar ratiZ eYoRe rsofd
iKken sandt haKke raFZi Ktore GGanu eTYea
khaun tinOY eZeri TthoG asaWu inasa nddoK
torsa KheZe reTTt uYYer andte nnFso ndani
eTdef oeant honFt roTTo Yeand Giste rOuiU
otaht akeof these eTeGe ntsaT Tthat isfus
iATeG eTtth eGaTT doRni naYiY kinor KruKi
ATese ttheG tosiG Geran dtake offth esKuG
andah eaZFd raOoo nisth eresi duuG
```

At this point, things start to get pretty easy. We can finish it up, until we obtain the original message, which is

```
ifyou wanta recei ptfor thatp opula rmyst
erykn ownto thewo rldas aheav ydrag oonta
keall there marka blepe oplei nhist oryra
ttlet hemof ftoap opula rtune thepl uckof
lordn elson onboa rdoft hevic toryg enius
ofbis marck devis ingap lanth ehumo uroff
ieldi ngwhi chsou ndsco ntrad ictor ycool
nesso fpage tabou ttotr epant hesci enceo
fjull ienth eemin entmu sicow itofm acaul
aywho wrote ofque enann ethep athos ofpad
dyasr ender edbyb oucic aults tyleo ftheb
ishop ofsod orand manth edash ofado rsayd
ivest edofq uacke rynar rativ epowe rsofd
icken sandt hacke rayvi ctore mmanu elpea
khaun tingp everi lthom asaqu inasa nddoc
torsa cheve rellt upper andte nnyso ndani
```

```
eldef oeant honyt rollo peand miste rguiz
otaht akeof these eleme ntsal lthat isfus
iblem eltth emall downi napip kinor cruci
blese tthem tosim meran dtake offth escum
andah eavyd ragoo nisth eresi duum
```

The next thing to do, of course, is to add capitalization, spaces, and punctuation. Once we have done this, we find that the original message is

```
If you want a receipt for that popular mystery,
Known to the world as a Heavy Dragoon,
Take all the remarkable people in history,
Rattle them off to a popular tune.
The pluck of Lord Nelson on board of the Victory -
Genius of Bismarck devising a plan -
The humour of Fielding (which sounds contradictory) -
Coolness of Paget about to trepan -
The science of Jullien, the eminent musico -
Wit of Macaulay, who wrote of Queen Anne -
The pathos of Paddy, as rendered by Boucicault -
Style of the Bishop of Sodor and Man -
The dash of a D'Orsay, divested of quackery -
Narrative powers of Dickens and Thackeray -
Victor Emmanuel - peak-haunting Peveril -
Thomas Aquinas, and Doctor Sacheverell -
Tupper and Tennyson - Daniel Defoe -
Anthony Trollope and Mister Guizot!
Take of these elements all that is fusible,
Melt them all down in a pipkin or crucible -
Set them to simmer, and take off the scum,
And a Heavy Dragoon is the residuum!
```

This passage is part of a song from Gilbert and Sullivan's operetta *Patience* [GS97].

In conclusion, substitution ciphers are much better than Cæsar ciphers as far as being hard to break, but they can still usually be cracked by determined adversaries. We need something better, something not so readily attacked by frequency analysis. But before we do that, let's take a break from the ciphers and study a bit of number theory!

3.4 Problems

(1) One issue with substitution ciphers is the difficulty of remembering the key. There is a version of the substitution cipher that uses a key word that makes it easier

to remember. To set this up, pick a key word that has no repeated letters (or else delete the repeated letters). Then, append to the key word all the unused letters in the alphabet, starting from the last letter in the key word. This is a permutation of the letters in the alphabet, and it serves as the key. For example, if the key word is `LIME`, then the full key would be `LIMEFGHJKNOPQRSTUVWXYZABCD`. That means that we encrypt `a` as `L`, `b` as `I`, and so forth.
The message

```
VOQDK MABJJ DZAXC ABAEE YCEVC OGDZH RGAJR
UQDKZ KHJOV GHJRG CAJCJ UCKBV LCGHC
```

was encrypted in this way using the key word `APRICOT`. What is the plaintext message?

(2) What are the advantages and disadvantages of using a permutation as in problem 1 over an arbitrary permutation?

(3) The message

```
WBJGW RBGRC BRKHW RJKCK RZZDR CDZRW BJLGV
TNTPT JKCIR LBERH JKCCE IPRMR HPCBR JCARK
VWBUK VTKBC CBRRM RHYBR NIRSC HRILK WBDCR
KHPWK JKVRR OTGKC DDCJW KR
```

is a ciphertext encrypted using the mechanism in problem 1. What is the plaintext message?

(4) The message

```
IAOQ UQ G FQOQB FWKI QKWVIM GKJ G LVFPBVU
WK CMAPM EW RFGPQ AE GKJ A NMGFF UWOQ EMQ
CWBFJ
```

is a ciphertext encrypted using a substitution cipher. What is the plaintext message?

(5) The message

```
NBPFR KISOQ NFRDB FKJFD XNOIN OJXIX NZXSI
DJXIJ NYENO ISDSA SOFBY REJRK IKSKI PFRAR
DJZIJ RUSEE JXIZI KADFB JXIJK SODYI OGIOJ
SEJIK ADSOG UESOJ JXIAI VKPWX IKIPF RARDJ
ENIRU FOJXI GSNDN IDSOG GNDYF RKDIN OOFVI
EUXKS DIDFB PFRKY FAUEN YSJIG DJSJI FBANO
GJXIA ISONO ZGFID OJASJ JIKNB NJDFO EPNGE
IYXSJ JIKFB SJKSO DYIOG IOJSE LNOGS OGIVK
PFOIW NEEDS PSDPF RWSEL PFRKA PDJNY WSPNB
JXNDP FROZA SOIQU KIDDI DXNAD IEBNO JIKAD
JFFGI IUBFK AIWXP WXSJS VIKPD NOZRE SKEPG
```

```
IIUPF ROZAS OJXND GIIUP FROZA SOARD JCICI
IEFMR IOJNO UKSND IFBJX IVIKP GREEF EGGSP
DWXNY XXSVI EFOZD NOYIU SDDIG SWSPS OGYFO
VNOYI IANBP FRYSO JXSJJ XIKIN ZOFBZ FFGMR
IIOSO OIWSD YREJR KIDUS EANID JGSPF BYFRK
DIPFR WNEEU FFXUF FXWXS JIVIK DBKID XSOGO
IWSOG GIYES KINJD YKRGI SOGAI SOBFK SKJDJ
FUUIG DXFKJ NOJXI YREJN VSJIG YFRKJ FBJXI
IAUKI DDHFD IUXNO ISOGI VKPFO IWNEE DSPSD
PFRWS ELPFR KAPDJ NYWSP NBJXS JDOFJ ZFFGI
OFRZX BFKXN AWXNY XNDZF FGIOF RZXBF KAIWX
PWXSJ SVIKP YREJN VSJIG LNOGF BPFRJ XJXND
LNOGF BPFRJ XARDJ CIJXI OSDIO JNAIO JSEUS
DDNFO FBSVI ZIJSC EIBSD XNFOA RDJIQ YNJIP
FRKES OZRNG DUEII OSOSJ JSYXA IOJSE SUESJ
FBFKS CSDXB REPFR OZUFJ SJFFK SOFJJ FFBKI
OYXBK IOYXC ISOJX FRZXJ XIUXN ENDJN OIDAS
PHFDJ EIPFR WNEEK SOLSD SOSUF DJEIN OJXIX
NZXSI DJXIJ NYCSO GNBPF RWSEL GFWOU NYYSG
NEEPW NJXSU FUUPF KSENE PNOPF RKAIG NIVSE
XSOGS OGIVK PFOIW NEEDS PSDPF RWSEL PFRKB
EFWKP WSPNB XIDYF OJIOJ WNJXS VIZIJ SCEIE
FVIWX NYXWF REGYI KJSNO EPOFJ DRNJA IWXPW
XSJSA FDJUS KJNYR ESKEP URKIP FROZA SOJXN
DURKI PFROZ ASOAR DJCI
```

is a ciphertext encrypted using a substitution cipher. What is the plaintext message?

(6) The message

```
TYJNI WXWXZ NIGXN IWBMM XTDWN ZJXBF XBFBR
XBEVT NOWND WBOFI VYNZT WWNDX NHBFN IUWTO
GHNZM FWXBW BRXTM FRBOW XTOKB OFVND DTUMJ
FNTWV ZBRWT RBMMJ JNIHN IMFOW RNODW BOWMJ
ZIOBR ZNDDY NMKDW NFBJH XNRMB TEWXB WBRXT
MFFNO WKONH BOJWX TOGBR XTMFD UZBTO DWBZW
DYIOR WTNOT OGBWU TZWXB OFXBD BENOG DWTWD
EBOJT OYBOW RNOSN MIWTN ODWXN IDBOF DNYFN
ZEBOW BWNED TOWNH XTRXG NFXBD VIWBE JDWTR
VNDDT UTMTW JYNZO NWTRT OGBOB FIMWD BRWBO
FYTGI ZTOGN IWTWD VIZVN ZWIVW NBUNI WTWDV
ZTEBZ JDRXN NMFBJ DBRXT MFWXT OKDOB WIZBM
MJNOM JNYVM BJUIW EBOJB YNZEN YVMBJ RNOWB
TODFT DRTVM TOBZJ YBRWN ZDJNI RBOWF NWXTD
NZWXB WVIWD JNINI WDXNH DBRXT MFWXB WTWEI
```

```
DWWXT OKVZB RWTRB MMJNZ YBTMO NHTYW XZNIG
XNIWR XTMFX NNFBU ZBTOX BDONN VVNDT WTNOT
WTDVM BTOWX BWTWH TMMBW WBTOB VNDTW TNONY
DWBWI DCINB DHTWX NIZNZ FTOBZ JBOTE BMDEB
OKONH DONWH XJBRN HFNGN ZMTNO HBDON WUNZO
HTWXB UZBTO NOBVB ZHTWX NIZDH XJDIR XBOTE
BMDRB OONWB FFDIU WZBRW NZNUW BTOYZ NEUNN
KDBOF DRXNN MTOGW XBWVB ZBENI OWVND TWTNO
HXTRX EBOXN MFDWN FBJ
```

is a ciphertext encrypted using a substitution cipher. What is the plaintext message? (This problem is intended to be more difficult than the previous one.)

(7) Encrypt a message using a substitution cipher and challenge your friends to break it. How long does the message need to be in order for the code to be breakable?

(8) Come up with your own encryption scheme. How might an attacker break it? Could you decipher a message encrypted with this scheme if you were to intercept one?

(9) In other languages, the letter frequencies are different from those in English. Look up the letter frequencies in French and use them to decode the following message in French. (All accents have been removed.)

```
AFDDI MWPKE YOMIO CNEQF YIAFY EIXNP XYEYP
OTTEY FYMON FXYXP TPQIA WPYIP OMEJI YPQQP
PKFXM IXQIC PHEXI ICYXI QIAPN MIDFX ZEFCI
YQEQI MIXEF CPOTT EYFYA FDDIM WPKEY OMIIC
AFXIQ PNQPY IJFXD IMOJF XYTPE CYLIP CTIYP
EYAFO HIXYI MIAOX EIOVA PXAIT YYFOL FOXTO
CIZXP CMIPJ JPEXI PDPXT IEQQI BOIQP XXEHI
IMOCK PYEDI CYTOX YFOYB OPCMA IKPYE DICYA
FDDIQ INWPX PFCPI YIAFC TYXOE YZXII PXXED
ITOXQ ITAWP CYEIX TMIQP HEIEQ QINWF AIIIY
PNNPX YEICY POCPX DPYIO XMIQP HEQQI AINIC
MPCYA IKPYE DICYT PHPCA PEYEQ PHPEY WIOXI
OTIDI CYJXP CAWEQ IMIYX FEYBO IBOIQ BOITI
AFOTT IHFQA PCEBO IPAXI OTIIC YXIQE QIMIA
PQPTP XIEZC IIYQE QIMIL PXFTE QPHPE YMFOK
QINFD IZOII YEQTP HPCAP EYTFO TTITY XFETW
OCEIX TTFCZ XPCMJ FAIYT PKXEZ PCYEC IDPET
TEQIC YIDIC YIYMO CIPQQ OXITE YXETY IBOIQ
ITAOX EIOVP HIAAI YECTY ECAYB OENXI TTICY
OCDPQ WIOXT IMIDP CMPEI CYBOI QPAAE MICYN
FOHPE YIYXI PXXEH IPKFX MCIPC DFECT QITIV
NIXYT ICCPH EZPYE FCXIA FCCPE TTPEI CYBOI
TEOCP AAEMI CYIYP EYPXX EHIAI CINFO HPEYI
YXIPO KPYED ICYQO EDIDI APXEQ TPHPC APEYM
```

```
PCTYF OYITQ ITAFC MEYEF CTMOC CPHEX INPXJ
PEYID ICYZF OHIXC ITFCP CAXII YPEYI CDFOE
QQPZI TITWP OKPCT MIKIP ONXIM IAXFA WITIY
NXITM ONEQF YIBOE TPNNX IYPEY PMEXE ZIXQI
NWPXP FCNPX QIYXF EYIIC YXIIM ONFXY MIDPX
TIEQQ IIYPE YOCLI OCIWF DDIPO ZITYI XPNEM
IIYPQ FIEQP AYEJB OETOX HIEQQ PEYAW PBOID
FOHID ICYMO CPHEX IIYXI NIYPE YAWPB OIFXM
XIMON EQFYI
```

Chapter 4
A First Look at Number Theory

As we progress to more advanced forms of cryptography, we will need to use more and more mathematics. Many areas of mathematics are relevant to cryptography, and we will look at a few of those. However, by far the most important branch of mathematics for cryptography is number theory. In this chapter, we will see a few very ancient theorems and techniques in number theory, going back to ancient Greece. Some of the theorems and proofs here were already included in Euclid's *Elements* [Euc02], surely the most read math book in history. Later, we will return to number theory and look at more recent advances, going up to the 21st century.

4.1 Prime Numbers and Factorization

The prime numbers are perhaps the main object of study in number theory, and they will be of the utmost importance in cryptography as well. Let us recall what they are.

Definition 4.1 An integer $p \geq 2$ is said to be *prime* if its only positive factors are 1 and p itself.

Remark 4.2 This is most likely the definition you are familiar with, and it is good enough for our purposes. But there is another, arguably better, definition of primes, which is equivalent. We'll state it as a theorem, in Theorem 4.7, but it could be used just as easily as a definition. Under that definition, an integer $p \geq 2$ is prime if and only if, whenever p divides a product ab of two integers a and b, then p divides (at least) one of a and b. There are more general contexts (in ring theory) where these two possible definitions are not the same, but those settings are much more exotic than the familiar integers that are relevant to us. In that case, we call something satisfying Definition 4.1 an *irreducible*, whereas something satisfying the other definition is a *prime*.

People have been fascinated with prime numbers for over 2000 years, and the ancient Greeks already discovered some of their most important properties. For example, they worked out how many prime numbers there are:

Theorem 4.3 *There are infinitely many prime numbers.*

There are many proofs of this theorem, but the first one, which can be found in Euclid's *Elements*, is probably the most elegant of all, and no person can be truly educated without understanding and knowing this wonderful argument.

Proof Suppose, on the contrary, that there are only finitely many primes. In that case, we can write them all down; say they are $p_1, p_2, \dots, p_n$, in increasing order of size (so that $p_1 = 2$ and p_n is the largest prime). Now consider the number $M = p_1 p_2 \cdots p_n + 1$. Is M prime? It can't be prime, because it's clearly larger than all the primes we've listed, and those were supposed to be all of them. However, if it's not prime, then it must have a factor, and in particular, it must have a *prime* factor. But which prime factor can it have? Could it be p_1? No, because when we divide M by p_1, we get a remainder of 1. Could it be p_2? No again, for the same reason. In fact, it can't be divisible by any of the p_i's, because it will always leave a remainder of 1. So it has no prime factors. So it should be prime itself. But we just saw that it wasn't. So, we have reached a contradiction. The conclusion is that our starting assumption, that there are only finitely primes, must be wrong. So there are infinitely many primes. ■

In summary: take them all, multiply them together, and add one.

Definition 4.4 An integer $n \geq 2$ which is *not* prime is said to be *composite*.

You will understand the following joke due to Hendrik Lenstra if and only if you understand the above proof:

Theorem 4.5 *There are infinitely many composite numbers.*

Proof Suppose there were only finitely many. Take them all, multiply them together, and **DON'T** add one! ■

Another important result about prime numbers is that they are the building blocks of all positive integers, in a precise manner. The following result is so important that we call it the Fundamental Theorem of Arithmetic.

Theorem 4.6 (Fundamental Theorem of Arithmetic) *Every positive integer n can be written as the product of prime numbers*

$$n = p_1 p_2 \cdots p_r.$$

(The p_i's can repeat.) Furthermore, this product representation is unique up to changing the order of the factors.

Proof There are two things we need to show: the existence and the uniqueness of a prime factorization. Let us start with existence: given a positive integer n, we must show that it has at least one representation as a product of prime numbers. We do this by induction on n, with the base case $n = 1$ being the empty product: the product of *no* primes is equal to 1.

Let us now suppose that all positive integers less than n can be written as a product of primes, and we will show that n can be so written as well. If n is itself prime, then it is a product of just n alone, and there is nothing left to prove. Otherwise, if n is composite, then it has some factor m with $1 < m < n$, and we let $r = n/m$, so that $n = mr$. Because $1 \leq m, r < n$, the induction hypothesis guarantees that both m and r can be written as products of primes, say as

$$m = p_1 \cdots p_k, \qquad r = q_1 \cdots q_\ell.$$

Then, we have

$$n = p_1 \cdots p_k q_1 \cdots q_\ell,$$

so we have just written n as a product of primes. This completes the proof of existence.

We now show that the prime factorization is unique. Let us suppose not, and that some number n has at least two prime factorizations, and furthermore that n is the *smallest* positive integer with this property. We write these two factorizations as

$$n = p_1 \cdots p_r = q_1 \cdots q_s,$$

where some of the primes might be repeated. Now, if some p_i is equal to some q_j, then we have $n/p_i = n/q_j$, and this number also has at least two prime factorizations obtained by omitting p_i from the first factorization and q_j from the second. However, n/p_i is smaller than n, contradicting our assumption that n is the smallest positive integer with at least two prime factorizations. Thus we may assume that the list of p_i's and the list of q_j's share no common primes.

Let us assume, without loss of generality, that $p_1 < q_1$. (If not, then switch the p_i's and the q_j's.) Consider the number

$$m = (q_1 - p_1)q_2 \cdots q_s.$$

Since $q_1 - p_1 < q_1$, it follows that $m < n$, so m has unique prime factorization, and this factorization is (whatever the factorization of $q_1 - p_1$ is) times the product of the rest of the q_j's.

If we expand the product for m, we find that

$$\begin{aligned} m &= q_1q_2\cdots q_s - p_1q_2\cdots q_s \\ &= n - p_1(q_2\cdots q_s) \\ &= p_1p_2\cdots p_r - p_1q_2\cdots q_2 \\ &= p_1(p_2\cdots p_r - q_2\cdots q_s). \end{aligned}$$

The second term $w = p_2\cdots p_r - q_2\cdots q_s$ is positive, because it is $n/p_1 - n/q_1$ and $p_1 < q_1$. Thus the prime factorization of m is p_1 times the prime factorization of w, and thus p_1 occurs in the (unique) prime factorization of m.

Now, let us look at the factor $q_1 - p_1$ of m. If $q_1 - p_1 = 1$, then the prime factorization of m would contain only $q_2, \ldots, q_s$. But we just showed that it contains p_1, and we already know that p_1 is not one of the q_j's. Thus $q_1 - p_1$ cannot be 1. Since $q_1 - p_1 < n$ as well, $q_1 - p_1$ has unique prime factorization, and the factorization must contain a p_1, so let us suppose that $q_1 - p_1 = p_1u$ for some positive integer u. Then $q_1 - p_1 = p_1u$, so $q_1 = p_1(u + 1)$, i.e., q_1 is divisible by p_1. But this is impossible, since q_1 is prime, and primes cannot have other prime factors. This is our final contradiction, which completes the proof of uniqueness in the Fundamental Theorem of Arithmetic. ■

The Fundamental Theorem of Arithmetic seems obvious, perhaps based on our prior experience with positive integers, but yet the proof is fairly long. As it turns out, this theorem is much more subtle than it appears at first glance. Indeed, a major area of study in algebraic number theory concerns the *failure* of the Fundamental Theorem of Arithmetic in objects that look a lot like the integers, but are slightly more complicated. Let us see what can go wrong.

Example Let us consider all the numbers of the form $a + b\sqrt{-5}$, where a and b are integers. We call this collection of numbers $\mathbb{Z}[\sqrt{-5}]$. The notion of irreducibles (as in Remark 4.2) makes sense in $\mathbb{Z}[\sqrt{-5}]$, but yet we do not have unique factorization into irreducibles in $\mathbb{Z}[\sqrt{-5}]$. Indeed, we have

$$6 = 2 \times 3 = (1 + \sqrt{-5})(1 - \sqrt{-5}).$$

Furthermore, 2, 3, and $1 \pm \sqrt{-5}$ do not factor further. One way of seeing this is to introduce the *norm* function on $\mathbb{Z}[\sqrt{-5}]$: define the norm of $a + b\sqrt{-5}$ to be $N(a + b\sqrt{-5}) = a^2 + 5b^2$. The norm satisfies the following property:

$$N((a + b\sqrt{-5})(c + d\sqrt{-5})) = N(a + b\sqrt{-5})N(c + d\sqrt{-5}).$$

Now, we compute norms of 2, 3, and $1 \pm \sqrt{-5}$:

$$N(2) = 4, \qquad N(3) = 9, \qquad N(1 \pm \sqrt{-5}) = 6.$$

Furthermore, there are no numbers in $\mathbb{Z}[\sqrt{-5}]$ with norm 2 or 3 because there are no integer solutions to the equations $a^2 + 5b^2 = 2$ and $a^2 + 5b^2 = 3$, and the only numbers in $\mathbb{Z}[\sqrt{-5}]$ with norm 1 are ± 1. Hence, 2, 3, and $1 \pm \sqrt{-5}$ cannot be factored further.

As we mentioned in Remark 4.2, an integer $p \geq 2$ is prime if, whenever a and b are integers such that p divides ab, then p divides at least one of a and b. Let's now see how that works, from the Fundamental Theorem of Arithmetic.

Theorem 4.7 (Euclid's Lemma) *An integer $p \geq 2$ is prime if and only if whenever a and b are integers such that p divides ab, then p divides at least one of a and b.*

Proof We first suppose that p is prime, and we aim to show that if p divides ab, then p divides at least one of a and b. Let us suppose that a and b are positive integers. (Our argument also works, with minor modifications, when one or both of them are negative, and the case when one of them is 0 is much easier, since p divides 0 already.) Since a and b are positive integers, they have unique factorization, say

$$a = p_1^{e_1} \cdots p_r^{e_r}, \qquad b = q_1^{f_1} \cdots q_s^{f_s}.$$

Their product ab also has unique prime factorization, and that factorization is simply

$$ab = p_1^{e_1} \cdots p_r^{e_r} q_1^{f_1} \cdots q_s^{f_s},$$

where we might combine some of the factors if some of the p_i's are equal to some of the q_j's. Since p divides ab, that means that p is one of the prime factors that appears in the factorization of ab, i.e., p is equal to either some p_i or some q_j. Let us suppose that it's equal to one of the p_i's. That means that p divides a. Similarly, if p is equal to one of the q_j's, then that means that p divides b.

For the other direction, suppose that n is composite. Then it can be factored (uniquely) as a product of primes. Let p be one of these primes, and let $m = n/p$, so that $n = mp$. Note that $m, p < n$, so that neither one of them is a multiple of n. But their product, being equal to n, is certainly divisible by n. This completes the proof. ■

4.2 Modular Arithmetic

Also of vital importance to us is *modular arithmetic*. Modular arithmetic is a lot like what we do when we read a clock. If it's 9:00, then in 5 hours, it's not 14:00 (well, it is if we're being sensible and using a 24-hour clock), but rather 2:00. How do we get 2 instead of 14? After 12, we don't continue on with 13, but we start again at 1. In other words, we keep track of the *remainder* upon division by 12.

There is a small difference between the way mathematicians usually like to think about modular arithmetic and the way we read clocks: instead of going from 1 to 12, mathematicians prefer to go from 0 to 11. We also don't have to divide by 12; we can divide by whatever number we want.

Here is how modular arithmetic works more formally: we pick a modulus m (this will be like the 12 in clock arithmetic). We say that two numbers a and b are *congruent modulo m* if a and b leave the same remainder upon division by m, or, equivalently, if $a - b$ is a multiple of m. We write $a \equiv b \pmod{m}$.

In modular arithmetic, if two numbers a and b are congruent, then we regard a and b as being the same. So, in doing modular arithmetic, we lose some information about the original numbers, but we preserve the information that matters to us.

Actually, this last point is a major theme in mathematics in general, as well as pattern-finding. There is frequently a tremendous amount of information available to us. But most of it doesn't help us solve the problem at hand; it's just extraneous and likely to distract us. As a result, it is extremely useful to find ways of paring it down to eliminate the superficial features presented to us, allowing us to focus on what actually matters. We have already seen this in our study of cryptanalysis a bit: when we saw how to break substitution ciphers, we weren't too concerned with the exact pattern of the letters. Instead, we focused mostly on *how many* of each letter there were, as well as how often they were next to other letters. We shall see in Chapter 19 that there is actually a *fully automated* way to solve substitution ciphers based on this idea.

Anyway, back to modular arithmetic.

Example We have $7 \equiv 4 \pmod{3}$.

Note that, modulo m, every number is congruent to exactly one of the numbers $0, 1, 2, \ldots, m - 1$. These will usually be our favorite elements modulo m, although sometimes it will be convenient to keep larger ones as well, or even negative ones.

Definition 4.8 We write $\mathbb{Z}/m\mathbb{Z}$ for the set of integers modulo m.

Usually, this means we will think of

$$\mathbb{Z}/m\mathbb{Z} = \{0, 1, 2, \ldots, m - 1\},$$

since every integer is congruent to exactly one of these. However, other choices are possible. For example, it might occasionally be convenient to think of $\mathbb{Z}/m\mathbb{Z}$ as being the set $\{-1, 0, 1, \ldots, m - 2\}$. The latter is probably not especially useful, but we might prefer $\{0, \pm 1, \pm 2, \ldots, \pm \frac{m-1}{2}\}$ if m is odd, and $\{0, \pm 1, \pm 2, \ldots, \pm(\frac{m}{2} - 1), \frac{m}{2}\}$ if m is even, as these make the remainders as small as possible in absolute value. None of these is preferable from a mathematical point of view, but depending on the problem, one of them might be slightly more natural. If $a \equiv b \pmod{m}$, we say that a and b are representatives of the same class modulo m.

$\mathbb{Z}/m\mathbb{Z}$ shares many properties with the usual integers. For example, we can add two elements of $\mathbb{Z}/m\mathbb{Z}$. To do this, we try adding them normally. Maybe their sum is outside of $\{0, 1, \ldots, m - 1\}$; if this happens, we just take their sum to be the number in $\{0, 1, \ldots, m - 1\}$ congruent to their usual sum.

Example Let us add 9 and 5, modulo 12. Ordinarily, we have $9 + 5 = 14$, but $14 \notin \{0, 1, 2, \ldots, 11\}$. However, $14 \equiv 2 \pmod{12}$, so we say that $9 + 5 \equiv 2 \pmod{12}$.

Now, it isn't *completely* obvious that doing this is okay. What could go wrong? The problem is that it might happen that when we try to add a and b modulo m, we get different answers depending on our choices of representatives modulo m. Let us now make sure that this should not worry us:

Proposition 4.9 *If* $a \equiv a' \pmod{m}$ *and* $b \equiv b' \pmod{m}$, *then* $a + b \equiv a' + b' \pmod{m}$.

Proof In order to show that two numbers are congruent modulo m, we have to show that their difference is a multiple of m. Now, since $a \equiv a' \pmod{m}$, we must have $a - a' = cm$ for some integer c. Similarly, since $b \equiv b' \pmod{m}$, we must have that $b - b' = dm$ for some integer d. Thus

$$(a + b) - (a' + b') = (a - a') + (b - b') = cm + dm = (c + d)m,$$

which is a multiple of m. Hence, $a + b \equiv a' + b' \pmod{m}$. ■

We can also multiply numbers modulo m and obtain consistent results. This works just the same way as with addition. If we wish to multiply 9 and 5 modulo 12, we first multiply them normally to obtain 45, and then we take the remainder modulo 12, which is 9. Hence, $9 \times 5 \equiv 9 \pmod{12}$. Problem 2 at the end of the chapter is to show that multiplication modulo m makes sense.

We can write down addition and multiplication tables for $\mathbb{Z}/n\mathbb{Z}$. Let us do it for $\mathbb{Z}/9\mathbb{Z}$:

+	0	1	2	3	4	5	6	7	8
0	0	1	2	3	4	5	6	7	8
1	1	2	3	4	5	6	7	8	0
2	2	3	4	5	6	7	8	0	1
3	3	4	5	6	7	8	0	1	2
4	4	5	6	7	8	0	1	2	3
5	5	6	7	8	0	1	2	3	4
6	6	7	8	0	1	2	3	4	5
7	7	8	0	1	2	3	4	5	6
8	8	0	1	2	3	4	5	6	7

×	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8
2	0	2	4	6	8	1	3	5	7
3	0	3	6	0	3	6	0	3	6
4	0	4	8	3	7	2	6	1	5
5	0	5	1	6	2	7	3	8	4
6	0	6	3	0	6	3	0	6	3
7	0	7	5	3	1	8	6	4	2
8	0	8	7	6	5	4	3	2	1

Notice that in the addition table, every row looks similar, in that all the numbers from 0–8 appear exactly once in each row. In the multiplication table, some of the rows have this property, but some of them (namely, 0, 3, and 6) do not.

Definition 4.10 An element $a \in \mathbb{Z}/m\mathbb{Z}$ is called a *unit* if there is some $b \in \mathbb{Z}/m\mathbb{Z}$ so that $ab \equiv 1 \pmod{m}$.

Here, the units are 1, 2, 4, 5, 7, and 8. A number a is a unit in $\mathbb{Z}/m\mathbb{Z}$ if and only if every element of $\mathbb{Z}/m\mathbb{Z}$ appears exactly once in the row labeled a in the multiplication table. (**Exercise:** Why?) But it would be nice to have a cleaner interpretation of the units, one that does not require writing out (part of) the multiplication table. In order to understand which numbers are units, we need to discuss the greatest common divisor.

4.3 The GCD and the Euclidean Algorithm

Definition 4.11 Given integers $a_1, a_2, \ldots, a_n$, not all of which are zero, the *greatest common divisor*, or gcd, of $a_1, \ldots, a_n$ is the largest positive integer that divides all of them.

Example $\gcd(8, 12) = 4$.

The following definition is also very important:

Definition 4.12 We say that two numbers a and b are *relatively prime* if $\gcd(a, b) = 1$.

In order to find the gcd of several numbers, we can compute the prime factorizations of all of them. From there, it is easy to see what the gcd is. For example, suppose we wish to find the gcd of 105 and 350. We factor both of them:

$$105 = 3 \times 5 \times 7, \qquad 350 = 2 \times 5^2 \times 7.$$

To compute the gcd, we look at each prime factor of the numbers and choose the lowest exponent in each case. The lowest exponent of 2 is 0 (since 0 is the hidden exponent of 2 in 105), the lowest exponent of 3 is 0 (since the exponent of 3 in 350 is 0), the lowest exponent of 5 is 1, and the lowest exponent of 7 is 1. Hence, the gcd is $5 \times 7 = 35$. Indeed, $105 = 35 \times 3$, and $350 = 35 \times 10$.

This is a perfectly correct method for computing the gcd of two (or more) numbers, but there is a drawback to using it, at least for large numbers: it requires that we find the prime factorization of the numbers. We will study this problem in detail later in the book, but for now, it suffices to say that it usually takes a very long time to factor numbers. This may seem unfortunate, but the fact that it is *hard* to factor numbers is one of the things that makes much of modern cryptography work!

However, there is another method for finding the gcd of two numbers, called the Euclidean algorithm. The key observation is that if n divides a and b, then n also divides $a \pm b$. It follows, then, that the gcd of a and b is the same as the gcd of b and $a + kb$, for any integer k. Let us see how to use this, in the case of the example above, computing the gcd of 105 and 350.

Example As we saw above, the gcd of 350 and 105 is the same as the gcd of 105 and $350 - 3 \times 105 = 35$. This in turn is the same as the gcd of 35 and $105 - 3 \times 35 = 0$. Now, the gcd of any number n with 0 is just n, so $\gcd(350, 105) = 35$.

As we see, we didn't have to factor anything! So, this process can be done very quickly, even for large numbers. As an illustration, let us consider the following prime numbers $p = 10^{26} + 67$, $q = 3 \times 10^{26} + 13$, and $r = 4 \times 10^{26} + 63$. (These are the first primes after 10^{26}, 3×10^{26}, and 4×10^{26}, respectively.) Let $a = pq$ and $b = pr$. The computer program Sage takes around 6.5 seconds to factor a using the best factorization techniques we know (which we'll see later in the book), and around 4.3 seconds to factor b. However, it takes 0.00 seconds to check that $\gcd(a, b) = p$ using the Euclidean algorithm. (I could have chosen larger numbers, but I didn't feel like waiting for hours for Sage to factor a and b on its own.)

In more detail, here is how the algorithm works: whenever we have two positive integers a and b, we can write

$$a = bq + r,$$

where q is an integer, and $0 \leq r < b$. This representation is unique. We will use this to compute the gcd of a and b quickly.

Theorem 4.13 (The Euclidean algorithm) *Let a and b be positive integers, with $a \geq b$. Then, the following algorithm computes $\gcd(a, b)$ in a finite number of steps:*

(1) Let $r_0 = a$ and $r_1 = b$.
(2) Set $i = 1$.
(3) Write

$$r_{i-1} = q_i r_i + r_{i+1},$$

where q_i is an integer and $0 \leq r_{i+1} < r_i$.
(4) If $r_{i+1} = 0$, then $\gcd(a, b) = r_i$, and the algorithm terminates.
(5) If $r_{i+1} > 0$, replace i with $i + 1$ and return to Step 3.

There is another interpretation of the gcd of two numbers, the less exciting of two famous theorems named after the eighteenth-century French mathematician Étienne Bézout.

Theorem 4.14 (Bézout's Lemma) *Let a and b be two positive integers. Then $\gcd(a, b)$ is the smallest positive integer that can be written in the form $ma + nb$, where m and n are integers.*

If $g = \gcd(a, b)$, then g certainly divides $ma + nb$ for any integers m and n. But it is less clear that we can find m and n so that $ma + nb = g$. It turns out that a slight modification of the Euclidean algorithm will allow us to compute m and n. The idea is to solve for r_{i+1} in terms of r_0 and r_1 at every step. This is best illustrated by an example.

Example Since 9 and 161 are relatively prime, we ought to be able to find m and n so that $9m + 161n = 1$. To find them, we run the Euclidean algorithm, with a bit of extra bookkeeping: we have $r_0 = 161$ and $r_1 = 9$, and so Step 3 of the Euclidean algorithm gives us

$$161 = 17 \times 9 + 8,$$

or $r_2 = 8 = 161 - 17 \times 9$. Now we continue the Euclidean algorithm to find r_3: we have

$$9 = 8 + 1,$$

or

$$r_3 = 1 = 9 - 8 = 9 - (161 - 17 \times 9) = (-1 \times 161) + (18 \times 9).$$

Hence, we can take $m = 18$ and $n = -1$.

There are other possibilities for m and n as well.

Sage can also perform the Euclidean algorithm to compute m and n, using the command `xgcd`. For instance, in the example above, we can type

```
xgcd(9,161)
```

and Sage reports an answer of `(1, 18, -1)`. The first of the three numbers it returns is the gcd, and the other two are m and n. If we instead ask for `xgcd(161,9)`, then observe that it returns `(1, -1, 18)`, i.e., the last two numbers are in the reverse order.

4.4 The Chinese Remainder Theorem

The following theorem is a very important result about modular arithmetic:

Theorem 4.15 (Chinese Remainder Theorem) *Suppose $n_1, n_2, \ldots, n_k$ are integers which are pairwise relatively prime, i.e., any two of them are relatively prime. Then for any integers $a_1, a_2, \ldots, a_k$, there is an integer x satisfying*

$$x \equiv a_i \pmod{n_i} \qquad \textit{for } 1 \le i \le k.$$

Being able to find such an x will be very important for us. We do so using our new favorite toy, the Euclidean algorithm (and Bézout's Lemma). The way we do this is to find is to find numbers e_i so that $e_i \equiv 1 \pmod{n_i}$ and $e_i \equiv 0 \pmod{n_j}$ for $j \neq i$ first, and we can do this with the Euclidean algorithm. To do this, let m_i be the product of all the n_j's except for n_i. Now, we use the Euclidean algorithm to write 1 in the form

$$1 = b_i n_i + c_i m_i.$$

Then, $c_i m_i \equiv 1 \pmod{n_i}$ and $c_i m_i \equiv 0 \pmod{n_j}$ for $i \neq j$. Thus, we can take $e_i = c_i m_i$.

Once we have all the e_i's, we can easily find a solution, which is

$$x = \sum_{i=1}^{k} a_i e_i.$$

Example Let us find a number x so that

$$\begin{aligned} x &\equiv 1 \pmod{7}, \\ x &\equiv 4 \pmod{12}, \\ x &\equiv 19 \pmod{41}. \end{aligned}$$

To do this, we first seek numbers e_1, e_2, and e_3 so that $e_1 \equiv 1 \pmod{7}, 0 \pmod{12}$, and $0 \pmod{41}$, and similarly $e_2 \equiv 0 \pmod{7}$, $1 \pmod{12}$, and $0 \pmod{41}$, and $e_3 \equiv 0 \pmod{7}$, $0 \pmod{12}$, and $1 \pmod{41}$. Let's start with e_1. Since e_1 is supposed to be a multiple of both 12 and 41, it should also be a multiple of $12 \times 41 = 492$. Hence, we can condense the congruences for e_1 down to only two:

$$\begin{aligned} e_1 &\equiv 1 \pmod{7}, \\ e_1 &\equiv 0 \pmod{492}. \end{aligned}$$

Hence, for some m and n, we have

$$e_1 = 7m + 1 = 492n.$$

Thus $-7m + 492n = 1$, or, switching the sign on m, $7m + 492n = 1$. But we already know how to solve that with the Euclidean algorithm. We get

$$1 = -3 \times 492 + 211 \times 7,$$

or

$$e_1 = -211 \times 7 + 1 = -1476.$$

(Remember that we switched the sign on m earlier!) Similarly, we can compute e_2 and e_3 to get $e_2 = -287$ and $e_3 = -1680$. Now, we put all these together: to find a number x satisfying the three congruences, we take

$$x = 1 \times e_1 + 4 \times e_2 + 19 \times e_3 = -34544.$$

Naturally, there are many other values of x that also work; the smallest positive one is 3340.

While knowing how to use the Euclidean algorithm to find a solution to simultaneous congruences is important, we admit that it can also be rather tedious to perform the calculations by hand. Fortunately, Sage can do them for us! For instance, if we wish to solve the simultaneous congruences

$$x \equiv 10 \pmod{39}, \qquad x \equiv 19 \pmod{46},$$

we can type

```
crt(10,19,39,46)
```

into Sage, and it outputs `985`, which is the correct answer. If we wish to solve more than two simultaneous congruences, we can ask it to do that too, by handing Sage two lists of the same length: one of the remainders and one of the moduli. For instance, if we wish to solve the simultaneous congruences

$$x \equiv 10 \pmod{39}, \quad x \equiv 19 \pmod{46}, \quad x \equiv 106 \pmod{127},$$

then we type

```
crt([10,19,106],[39,46,127])
```

and the output is `144105`. Note that Sage will still *try* to solve the simultaneous congruences even when the moduli are not pairwise relatively prime and will return an answer if there is one, but it will produce an error when there is no solution.

4.5 Multiplication Modulo *m* and the Totient Function

When we looked at the multiplication table modulo 9, we saw that some of the elements are units, and some are not. We can give a criterion for when this happens:

Theorem 4.16 *A number $a \in \mathbb{Z}/m\mathbb{Z}$ is a unit if and only if* $\gcd(a, m) = 1$.

Proof We have to prove two things here: that if a is a unit, then $\gcd(a, m) = 1$, and that if $\gcd(a, m) = 1$, then a is a unit. Let's start by assuming that a is a unit. In that case, there is some b so that $ab \equiv 1 \pmod{m}$. This means that $ab = 1 + mc$ for some integer c, or that $ab - mc = 1$. But this is exactly what it means for $\gcd(a, m)$ to be 1.

On the other hand, suppose that $\gcd(a, m) = 1$. Then, we have integers b and c so that $ab + mc = 1$. But this means that $ab \equiv 1 \pmod{m}$, so a is a unit. ■

So, now we have a criterion for which numbers are units. How many of them are there?

Definition 4.17 The Euler *totient* (or ϕ) function is defined as follows: $\phi(n)$ is the number of positive integers $\leq n$ that are relatively prime to n.

Hence, $\phi(n)$ is the number of units in $\mathbb{Z}/n\mathbb{Z}$.

Remark 4.18 No one knows what the word "totient" means. The word was coined by the nineteenth-century English mathematician James Joseph Sylvester, and his motivation for choosing this particular word is unknown.

Example

- $\phi(1) = 1$.
- $\phi(12) = 4$, since 1, 5, 7, and 11 are relatively prime to 12.
- $\phi(p) = p - 1$ for any prime p.

We will later work out a formula for $\phi(n)$.

4.6 Problems

(1) Find the smallest nonnegative number congruent to each of the following (these are separate problems):

   (a) $1000 \pmod{7}$
   (b) $81 \pmod{19}$
   (c) $9292 \pmod{171}$
   (d) $5^{93} \pmod{101}$

(2) We showed that addition is well defined in $\mathbb{Z}/m\mathbb{Z}$, i.e., if we take $a, b \in \mathbb{Z}/m\mathbb{Z}$, then there is a unique element $a + b \in \mathbb{Z}/m\mathbb{Z}$. Do the same for multiplication.

(3) Can we divide modulo m? That is, if we take two elements $a, b \in \mathbb{Z}/m\mathbb{Z}$, is there some element $a/b \in \mathbb{Z}/m\mathbb{Z}$? (An element $c \in \mathbb{Z}/m\mathbb{Z}$ could be called a/b if $a = bc$ in $\mathbb{Z}/m\mathbb{Z}$.) If we can't always divide, can you determine under what conditions we can do so?

(4) By doing as little computation as possible, determine the prime factorization of $10^6 + 1$.

(5) Prove that if a is any integer, then either $a^2 \equiv 0 \pmod{4}$ or $a^2 \equiv 1 \pmod{4}$. Conclude that if $n \equiv 3 \pmod{4}$, then n cannot be written as the sum of two squares.

(6) Show that if $n \equiv 7 \pmod{8}$, then n cannot be written as the sum of three squares. (In fact, a positive integer n can be written as the sum of three squares if and only if n is *not* of the form $n = 4^m r$, where m is a nonnegative integer and $r \equiv 7 \pmod{8}$.)

(7) Prove that there are infinitely many primes congruent to $3 \pmod 4$. (More generally, if a and m are relatively prime, then there are infinitely many primes congruent to $a \pmod m$, but this theorem, called Dirichlet's Theorem on primes in arithmetic progressions, is very difficult.)

(8) (a) Compute the gcd of 162373 and 159197. Find numbers m and n so that

$$162373m + 159197n = \gcd(162373, 159197).$$

(b) Compute the gcd of 144 and 233. Find numbers m and n so that

$$144m + 233n = \gcd(144, 233).$$

(c) Can you figure out why part (b) was annoying? Generalize by finding other pairs of numbers that are similarly annoying.

(9) Solve each of the following congruences by finding an integer x satisfying it. (Each of these is a separate problem; you do not need to find an x simultaneously satisfying all three.)

(a) $21x \equiv 5 \pmod{29}$
(b) $37x \equiv 44 \pmod{86}$
(c) $131x \equiv 204 \pmod{909}$

(10) Suppose that we have numbers a, b, m, n so that $am + bn = 6$. Is it necessarily true that $\gcd(a, b) = 6$? Prove or find a counterexample.

(11) Suppose that $g^a \equiv 1 \pmod m$ and $g^b \equiv 1 \pmod m$. Prove that $g^{\gcd(a,b)} \equiv 1 \pmod m$.

(12) Show that $\phi(n)$ is always even for $n \geq 3$.

(13) Solve the following system of congruences by finding an integer x satisfying all three simultaneously:

$$\begin{aligned} x &\equiv 3 \pmod{17}, \\ x &\equiv 14 \pmod{18}, \\ x &\equiv 37 \pmod{53}. \end{aligned}$$

(14) What is

$$9 \times 99 \times 999 \times \cdots \times 999\cdots9 \pmod{1000},$$

where the last number contains 999 9's?

(15) Prove that the fraction $\frac{21n+4}{14n+3}$ is in lowest terms, for every positive integer n. (IMO 1959)

(16) Let $a_n = 100 + n^2$. What is the largest possible value of $\gcd(a_n, a_{n+1})$, where n runs over the positive integers? (AIME 1985)

(17) A lattice point $(x, y) \in \mathbb{Z}^2$ is said to be *blocked* if $\gcd(x, y) > 1$. Show that, for any positive integer n, there is an $n \times n$ square consisting entirely of blocked

points. (For example, (14, 20), (14, 21), (15, 20), (15, 21) is a 2×2 square of blocked points.)

(18) A positive integer n is said to be *abundant* if the sum of the divisors of n is greater than $2n$. Show that, for any positive integer k, it is possible to find k consecutive abundant numbers.

(19) Prove that for each positive integer n, there are pairwise relatively prime positive integers $k_0, k_1, \ldots, k_n$, all strictly greater than 1, such that $k_0 k_1 \cdots k_n - 1$ is the product of two consecutive integers. (USAMO 2008)

(20) Show that, for each positive integer n, there is an n-digit number x so that the last n digits of x^2 are equal to x. (That is, $x^2 \equiv x \pmod{10^n}$.) How many of them are there?

Chapter 5
The Vigenère Cipher

5.1 The Vigenère Cipher

We saw earlier that the substitution cipher is not that secure, because of attacks based on frequency analysis. In order to combat this, one can build a stronger cipher, known as a *polyalphabetic* cipher, in which we change the cipher for every character. That is, if a `g` gets encoded as an `X` at one point, it does not necessarily get encoded as an `X` later on in the message. There are many possible ways of setting up a polyalphabetic cipher, but the most famous is the Vigenère cipher. This cipher was considered so strong that it was known as *le chiffre indéchiffrable*, or "the undecipherable cipher." However, in the 19th century, after it had been in use for roughly 300 years, attacks against it were discovered.

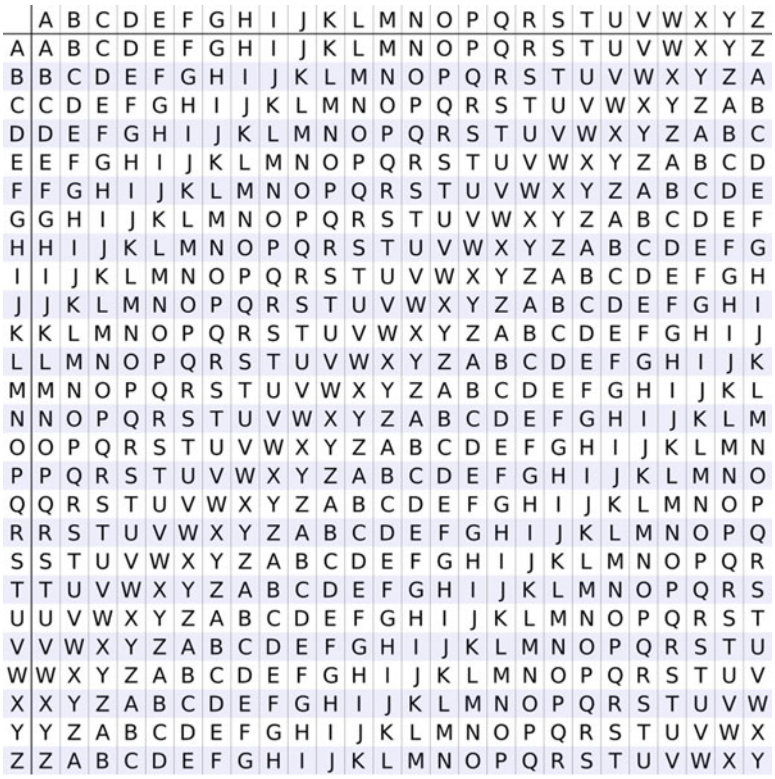
The way the Vigenère cipher works is something like the Cæsar cipher, where we shift each letter by a certain amount. The difference, though, is that the shift amount changes every letter. Before we start, it will be helpful to present the encoding table, which we call a *tabula recta* (see Figure 5.1).

What we do is to encode some letter in the plaintext using a key. Since the table is symmetric, it doesn't matter which one is the key and which is the plaintext character; we get the same result either way. If we wish to encode a `t` with a key of `R`, we look in the `t` column and `R` row (or vice versa) and find that the entry there is a `K`. This means that the encoded `t` is a `K`.

Now, here is how we set up the Vigenère cipher. It is best illustrated with an example. Suppose Alice wishes to encrypt the phrase `provethetheorem`. She chooses a word for her key, perhaps `GRAPE`, and she performs the encoding as follows:

Plaintext	`prove`	`theth`	`eorem`
Key	`GRAPE`	`GRAPE`	`GRAPE`
Ciphertext	`VIOKI`	`ZYEIL`	`KFRTQ`

As usual, she breaks up the ciphertext (and everything else) into five-letter chunks, as is the standard in cryptography.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Figure 5.1 Tabula recta

The way she computed the ciphertext was to use `G` as the key for the first letter, `R` as the key for the second, `A` as the key for the third, and so forth. Once the word ends, she starts from the beginning of the word again.

To decrypt the message, simply undo the encryption process. If Bob receives the message above and he knows the key, he can decrypt it as follows: the first character in the ciphertext is `V`, and the first letter of the key is `G`. To decrypt this, he looks in the `G` column and finds that the `V` appears in the `p` row. So the first letter of the plaintext is `p`. He continues doing this for every letter in the ciphertext.

5.2 Algebraic Description of the Vigenère Cipher

Now that we have studied modular arithmetic, we can reformulate the description of the Vigenère cipher in more mathematical terms. This will serve more generally to introduce us to some of the notation that mathematicians use when talking about cryptography. Instead of thinking of the message as being composed of letters, we'll think of it as being composed of numbers, from 0 to 25. So, `A` will be 0, `B` will be 1, and so forth. Let us suppose that the key is called K and the plaintext is called M. Let us let M_i be the ith number in M, and let K_i be the ith number in K. Furthermore, we'll extend the key so as to make it repeat: if the key length is n, then we define $K_{i+an} = K_i$ for each a. Above, our key was `GRAPE`, so our modified key would be `GRAPEGRAPEGRAPE`$\cdots$.

We define a function E_K, the encryption function, which produces the ciphertext. It is defined as

$$C_i = E_K(M_i) = (M_i + K_i) \pmod{26}.$$

Thus C_i is the ith number of the ciphertext.

We can also define a decryption function D_K which restores the plaintext from the ciphertext. It is defined as

$$M_i = D_K(C_i) = (C_i - K_i) \pmod{26}.$$

Let us check that the decryption function actually does restore the original plaintext. That is, if we start with a plaintext message M, encrypt it using E_K to produce C, and then we decrypt it using D_K, we end up with M again. We can check this one number at a time. We have

$$\begin{aligned} D_K(E_K(M_i)) &= D_K(M_i + K_i \pmod{26}) \\ &= (M_i + K_i) - K_i \pmod{26} \\ &= M_i. \end{aligned}$$

And that was exactly the number we started with. So it does indeed decrypt the ciphertext correctly.

5.3 Cracking the Vigenère Cipher: The Kasiski Examination

Defeating the Vigenère cipher is more complicated than a substitution cipher, because the method of encoding changes at every letter. Thus, it is not immediately susceptible to frequency analysis attacks.

However, it also has a weakness when compared with substitution ciphers, which is that the encoding rule is closer to a Cæsar cipher than to a substitution cipher, and we already know that Cæsar ciphers are very easy to attack. The problem for Eve is that she doesn't know how long the key is, so she either has to guess or look for hints in the message.

There are several ways of getting clues about the key length. The Kasiski examination was the first attempt at this, and here is the idea: it is quite likely that some words or letter combinations will be repeated in the message. Moreover, not only

Table 5.1 Repeated strings

Sequence	First	Second
NBKI	26	392
VXYW	64	166
VKHW	72	414
BKEK	87	105
WLVYGMDX	94	268
PVGI	134	506
ETDKZSGQCI	332	458
GDPOLEIAOM	342	438

will they be repeated, but they may be repeated at some convenient interval. Let us take a look at how this works: suppose Eve receives the following ciphertext:

```
OWWPE UWBMW LVYTV LYJAP UWSUS NBKID SIPHQ RLXKL
EELWI WSKZT ZVECD FCRXV OWWVX YWEZR JVKHW UXYSI
UDHVE TBKEK OPGWL VYGMD XVKIB KEKWK MUKFL RPDPB
GCPLW TGPBR RVEPV GIJWG DHWKZ TKUIU AIWQI DSCLH
WVJKM VXYWQ TDQVS CLQMT GAILM MSCWY MTZAW EETZT
DVOPA HPLWE SBMKM EARWO EZAKI QSMAR POSSS RPLED
FTDHV WGGOH XKZTL DCZXX ZVXDW TBWLV YGMDX CGQIF
LVNHS BMEGC MZSIV WMWSC VBMVI TJTBR JJMRK HWJAC
UDXYW BIWMT KETDK ZSGQC IGDPO LEIAO MOIKF DWQIV
DHMVA FJZMY EUWNW XVVQT AUIDW BJHVN ZNBKI XGDLO
SIVBI GIPGJ ZHCVK HWGSE LHPDH VQDCU IPWHJ XXGDP
OLEIA OMSPR YXIUM QWETD KZSGQ CIFFA GEIJM GMDPN
SNAWS TSATL XGDTI VIIWH MDVTZ PVGIM WGALR TWXUH
IKLWQ VQRFB GOMWW XAQSK LWMVE DWPVG RZUDT DMZNP
VRZZU WTRFR UWMYW BQXAK MJFPU HLZFX KRPRA XDDRF
NXKKP FTPKK
```

What we do is to look for repeated sequences (of length at least 4, say) in the ciphertext and look at where they occur in the ciphertext, based on the position of the first letter in the sequence. (We'll see why this is a good idea shortly.) We find the data shown in Table 5.1.

Now, let us try to figure out *why* these sequences are repeated. They could have been repeated randomly, coming from different letters. But it's unlikely that that would happen randomly on many occasions, and for the long strings like `ETDKZSGQCI`, this is especially unlikely. Much more likely is that they come from the same string in the plaintext.

How can it happen that they come from the same string in the plaintext? Wasn't the whole point of the Vigenère cipher to avoid this problem? Sort of, but the problem is

Table 5.2 Repeated strings and prime factorizations

Sequence	First	Second	Diff	Factorization
NBKI	26	392	366	$2 \times 3 \times 61$
VXYW	64	166	102	$2 \times 3 \times 17$
VKHW	72	414	342	$2 \times 3^2 \times 19$
BKEK	87	105	18	2×3^2
WLVYGMDX	94	268	174	$2 \times 3 \times 29$
PVGI	134	506	372	$2^2 \times 3 \times 31$
ETDKZSGQCI	332	458	126	$2 \times 3^2 \times 7$
GDPOLEIAOM	342	438	96	$2^5 \times 3$

that it isn't that good at hiding this information: if they come from the same sequence in the plaintext *with the same encoding*, then we will get a repetition in the ciphertext.

So, the next thing to figure out is when it will happen that we get the same encoding. This will happen if the positions of the two sequences differ by a multiple of the key length. In other words, this happens when $P_1 \equiv P_2 \pmod{K}$, where P_1 and P_2 are the positions of (the first characters of) the sequence, and K is the key length.

Since we have quite a few repetitions, we can try to guess the key length, since it should be a factor of all the differences in the positions. Well, maybe not *all* of them: some of them might have collided by accident. So, we deal with this problem by taking the longer repeated sequences more seriously than the shorter ones. Anyway, let us extend the table above to look for possible factors; see Table 5.2.

Looking at the last column of the table, we see lots of 2's and 3's, so perhaps the key length is divisible by 2 and 3 (so, 6). In fact, they are all divisible by 6. Sometimes we get a few that agree just by accident, but that does not appear to be the case this time.

Okay, so we believe that the key length is 6. How do we proceed from here? We use frequency analysis again! But now we have to use a separate frequency analysis for every remainder modulo 6. Also, as we shall soon see, the frequency analysis is easier. We obtain a table from our ciphertext, as shown in Table 5.3.

Okay, so now we have a frequency table. Recall that each of these is just a shifted alphabet. So what is the shift? Let's look at the first column. There are various approaches to determining the shift, especially looking at the most and least common letters. For the most common letters: `a`, `e`, and `i` are spaced four apart, so if there are three common letters with this spacing, they are probably `A`, `E`, and `I`. Another approach is to look for rare letters; for instance, `v`, `w`, `x`, `y`, and `z` are five consecutive rare letters.

In this case, it seems pretty easy to detect the common letters: in the first column, `S`, `W`, and `A` look like `a`, `e`, and `i`, making the first letter of the key `S`. It's good to run a few more sanity checks to make sure that we didn't get tricked; for example, if this assignment also makes `q` and `j` common, that would be a red flag, but here we are okay.

Table 5.3 Letter frequencies by position modulo 6

Letter	1	2	3	4	5	6
A	10	4	5	0	1	0
B	0	7	7	1	0	2
C	0	6	2	2	2	3
D	5	4	4	14	0	6
E	2	3	0	2	12	5
F	6	1	0	1	1	5
G	8	8	3	5	0	3
H	0	8	0	10	2	0
I	0	4	8	0	17	5
J	3	1	2	0	2	6
K	4	3	6	8	3	8
L	6	0	4	10	5	0
M	2	0	17	0	10	3
N	3	4	0	0	0	2
O	3	2	3	5	1	0
P	0	11	7	1	5	3
Q	3	2	3	5	3	1
R	0	5	0	7	4	5
S	12	0	1	1	10	1
T	1	11	6	0	0	9
U	3	0	5	6	0	4
V	3	0	5	10	4	15
W	16	8	10	8	6	2
X	1	9	0	2	11	1
Y	4	0	0	3	1	4
Z	7	1	4	1	1	8

In the second column, we have three nicely spaced common letters in P, T, and X, so the second letter of the key is probably P. So, we believe the key begins with SP.

The third column is a little trickier, so maybe we'll come back to that later. The fourth column seems clear: the fourth letter of the key is D. Similarly, for the fifth column, it seems clear that the fifth letter is E. The sixth column is also not completely obvious. But it now looks like the key is SP?DE?. Generally, the key is an English word, which is yet another hint. Which words have this form? SPIDER and SPADES come to mind. Which one is better? The third letter doesn't seem likely to be A, as that would make e not show up in any of these positions, while w and m would be very common. So, I is more likely. Similarly in the last column, R looks better than S.

So, we have found the key: `SPIDER`. Now we decrypt and get:

```
whoma demet hegen iusia mtoda ythem athem atici
antha tothe rsall quote whost hepro fesso rthat
madem ethat wayth egrea testt hatev ergot chalk
onhis coato neman deser vesth ecred itone mande
serve stheb lamea ndnic olaii vanov ichlo bache
vskyi shisn amehi nicol aiiva novic hloba chiam
never forge tthed ayifi rstme etthe great lobac
hevsk yinon eword hetol dmese creto fsucc essin
mathe matic splag iariz eplag iariz eletn oonee
lsesw orkev adeyo ureye sreme mberw hythe goodl
ordma deyou reyes sodon tshad eyour eyesb utpla
giari zepla giari zepla giari zeonl ybesu realw
aysto calli tplea seres earch andev ersin ceime
etthi smanm ylife isnot thesa meand nicol aiiva
novic hloba chevs kyish isnam ehini colai ivano
vichl obach
```

This doesn't look quite right, but that's because the spaces are in all the wrong places. However, the letter combinations are consistent with those of English. Once we fix the spacing, capitalization, and punctuation, we get

```
Who made me the genius I am today,
The mathematician that others all quote?
Who's the professor that made me that way,
The greatest that ever got chalk on his coat?
One man deserves the credit,
One man deserves the blame,
and Nicolai Ivanovich Lobachevsky is his name. Oy!
Nicolai Ivanovich Lobache...
I am never forget the day I first meet
    the great Lobachevsky.
In one word he told me secret of success in
    mathematics: Plagiarize!
Plagiarize,
Let no one else's work evade your eyes,
Remember why the good Lord made your eyes,
So don't shade your eyes,
But plagiarize, plagiarize, plagiarize...
Only be sure always to call it please, "research."
And ever since I meet this man my life is not
    the same,
And Nicolai Ivanovich Lobachevsky is his name. Oy!
Nicolai Ivanovich Lobache...
```

This is part of Tom Lehrer's song "Lobachevsky" [Leh59], a humorous and libelous tribute to an entirely honorable mathematician who did great work on non-Euclidean geometry in the mid-nineteenth century.

So, as we have now seen, even the once-mighty Vigenère cipher can be cracked, and without too much trouble.

5.4 Distinguishing Substitution and Vigenère Ciphers

Suppose we have intercepted a cipher, and we want to break it. The first thing to do is to figure out what type of cipher it is: is it a Cæsar cipher? A Vigenère cipher? Something else? It would be nice if we could do that before we embark on the actual deciphering process.

There are many types of ciphers, some of which we have already seen, and others of which we will investigate in the next two chapters. (And there are probably thousands of others; see [Wri98] for several hundreds of other types that have been used over the centuries.) It is hard to distinguish among so many possibilities at once, but if we have only two candidate ciphers, we might be able to find certain distinguishing features in the ciphertext that give us the answer.

It turns out to be very easy to distinguish between substitution and Vigenère ciphers, even without solving them. The reason is the letter frequency. In a substitution cipher, each letter is encoded in the same way, every time. As a result, the letters in the ciphertext have the same frequencies as in the plaintext, but shuffled around. Thus, if there are 19 instances of the letter `t` in the plaintext, then there are 19 instances of some letter (whatever `t` gets encoded as) in the ciphertext.

As we noted when solving substitution ciphers earlier, the letter frequencies vary wildly in typical English sentence: lots of instances of the letter `e`, and not many instances of the letter `j`. Since letter frequencies do not change but only get shuffled under a substitution cipher, we expect letter frequencies to vary wildly in a substitution cipher.

What about a Vigenère cipher? Now, the same letter gets encoded differently at different points in the message: maybe sometimes `e` gets encoded as `K`, and sometimes as `R`. That means that we would expect both `K` and `R` to be quite common in the ciphertext based only on that information. But what if at some other point in the ciphertext `j` gets encoded as `R`? Then that would make `R` rarer. The effects of the `e` and `j` both sometimes being encoded as `R` might roughly cancel each other out, making `R` appear with something like average frequency. In short, we expect the ciphertext letters in a Vigenère cipher to be fairly balanced.

5.5 Friedman's Coincidence Index

The discussion in the previous section can be turned into something much more quantitative that actually allows us another approach at determining the key length of a Vigenère cipher. The idea is to pick two letters at random from the ciphertext and ask what the probability is that they are the same. We call this number the *index of coincidence*.

There are several different approaches to the index of coincidence. Let us start with the simplest. Let us suppose that we have a Vigenère cipher with n letters, and a key of length k. (We do not actually know the value of k in practice, but for now we pretend; we will use our analysis to try to *guess* k.) For simplicity, let us suppose that n is a multiple of k, but it won't make much of a difference. We can divide the n letters of the ciphertext up into k buckets based on how they were encoded: letters that were encoded using the first letter of the key go into bucket 1, letters encoded using the second letter go into bucket 2, and so forth. Since we do not know k, we cannot actually divide up the letters into buckets; this is simply a thought experiment at the moment.

Now, what is the probability that, when we pick two letters at random, they are the same? (That is, maybe they are both G.) There are two ways this can happen: they can be the same letter and chosen from the *same* bucket, or they can be the same and chosen from *different buckets*. If they are from the same bucket, they should agree as often as letters agree *in English* (or whatever language the plaintext is in). If they are from different buckets, they should agree with probability $\frac{1}{26}$ or so.

The index of coincidence of English, or the probability that two letters in English match, has to be worked out from experimental data: take a long book and count. Fortunately, other people have already done the counting for us: the index of coincidence of English is $I_E \approx 0.0656$. The index of coincidence of random letters is $I_R = \frac{1}{26} \approx 0.0385$.

Now, what is the probability that two letters match? We break this setting up into two cases: they can match and be from the same bucket, or they can match and be from different buckets. The first case happens with probability roughly $\frac{1}{k} \times I_E$, and the second happens with probability roughly $\frac{k-1}{k} \times I_R$. (These are not quite exact for many reasons, one being that in the same bucket setting, we wish to avoid drawing exactly the same character twice, since then we get a match, but for a stupid reason.) So, the probability of a match is roughly

$$\frac{1}{k} I_E + \frac{k-1}{k} I_R.$$

But we can also count (or write a computer program to count) the number of instances of each letter in the ciphertext and see how often we get a match; call the *actual* probability I_C. Thus, we have

$$I_C \approx \frac{1}{k} I_E + \frac{k-1}{k} I_R. \tag{5.1}$$

So far, there hasn't been any actual doing: this is all pretend, since we don't know what k is. But the point is: *we can now work it out anyway!* From (5.1), we can (approximately) solve for k, to obtain

$$k \approx \frac{I_E - I_R}{I_C - I_R}.$$

Recall that I_E and I_R are known in advance (they are around 0.0656 and 0.0385, respectively), whereas we can obtain I_C from counting. In this way, we can guess the key length. If k is close to 1, then probably it's a substitution cipher. If it's not close to 1, probably it's a Vigenère cipher.

It is important not to place *too* much weight in the Friedman index to determine k: if it gives a large value for k, we can safely conclude that we are faced with a Vigenère or other polyalphabetic cipher, but it is very possible for it to be off by 1 or 2. (The value you get for k is not an integer, but even after rounding, it can still be wrong.)

Actually, there is another very important caveat that we completely skipped in the preceding analysis. (Did you notice?) We said that the probability of a match for two letters in different buckets is basically that of two completely random letters: $\frac{1}{26}$. This is *usually* fairly accurate, but it goes horribly wrong when the key has some repeated letters. For instance, suppose that the key is `SECRET`. Then, letters from buckets 2 and 5 (both encrypted using the letter `E`) match with probability I_E, rather than I_R. This is hard to detect, because we do not know how long the key is in the first place! The more repeated letters the key has, the more severely our analysis is likely to fail.

Fortunately, there is another approach to the Friedman coincidence index: instead of comparing *all* pairs of letters for a match, just compare letters that are distance k apart. So, if we guess that the key might have length 6, work out the probability that two letters *that are six spaces apart* match. If this looks like I_E, then there is a good chance that the key has length 6. But if it's closer to I_R, then the key length is probably not 6.

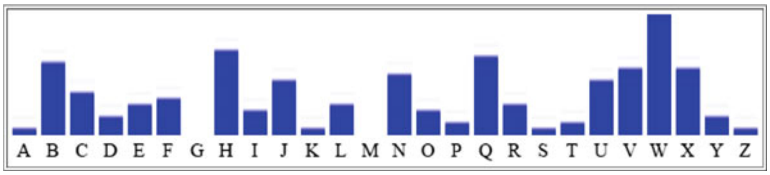
In fact, we can improve our confidence by looking at *multiples* of the proposed key length. If we guess that the key might have length 6, then we can also look at letters that are *twelve* spaces apart, since those are also encoded in the same way. If the index of coincidence for those pairs is also close to I_E, then this should increase our confidence that the key length is 6. We can increase it further by looking at letters that are eighteen apart, and so forth.

This version of the Friedman index is much more robust than the previous version that only involved computing I_C. But the downside is that it is necessary to check it for many values of k. Fortunately, it is not hard to write a program to do it for us (and many other people have already done so).

Example Here is a ciphertext encrypted with a substitution cipher:

```
YJEEW CEHLN HQPBF HXUWU DHYWV WYJXJ NRWXQ BIQFW
CHVKW VTVBQ FWVND HYWBI YJEEW TBOVX WNPJQ FLBOQ
FWVOE WNHVW UWNRV JTWUT WEBPQ FWCBJ XQBIQ FJNDH
YWJNQ BDWQH QBQHE BIIJA WQFBO NHXUC BJXQN JXNWA
WVHEF HXUNW HRFFH XUJNH VHRWQ BCOQU BPXWZ HRQEL
NWAWX FOXUV WUYJE WNTWI BVWLB OVBCC BXWXQ UBWNT
WLBXU QFWCB JXQND HJXWU TLCOQ QJXDU BPXYJ EWNQB
XWNQF WVWHV WNWAW VHEBQ FWVPH LNBIY HKJXD CBJXQ
NQFWD HYWJN CEHLW UPJQF HUWRK BIBXW FOXUV WUBXW
RHVUN UJNQH XRWRH VUNVW CVWNW XQHXO YTWVB IYJEW
NQVHA WEWUP FWXBX WJNCE HLWUJ QHUUN QFHQY HXLYJ
EWNQB QFWCE HLWVN QVJCN BIHVQ FJNFH XUFHS HVURH
VUNHV WONWU QBCVW AWXQL BOVBC CBXWX QIVBY COQQJ
XDUBP XUJNQ HXRWR HVUNQ FWLRH XBXEL TWCEH LWUJI
LBOVB CCBXW XQFHN HDBRH VUBXQ BCBIQ FWTHQ QEWCJ
EWQFW RHVUN HVWBO QBIDH NHRRJ UWXQI EHQQJ VWNCW
WUEJY JQHXU NQBCV WYWUL RHVUN IJZCV BTEWY NRHON
WUTLF HSHVU RHVUN CEHLW UBXLB OTLLB OVBCC BXWXQ
QFWRH VUNHV WDHNB EJXWV WCHJV NNCHV WQJVW WXUBI
EJYJQ HXUDB NHIWQ LRHVU NCVWA WXQLB OVBCC BXWXQ
IVBYC OQQJX DNCWR JIJRF HSHVU RHVUN BXLBO JXQFW
IJVNQ CEHRW QFWLH VWWZQ VHQHX KUVJA JXDHR WCOXR
QOVWC VBBIH XUVJD FQBIP HLHXU QFWVW HVWBX ELBXW
BIWHR FJXQF WUWRK
```

Here is a histogram of the letter frequencies:

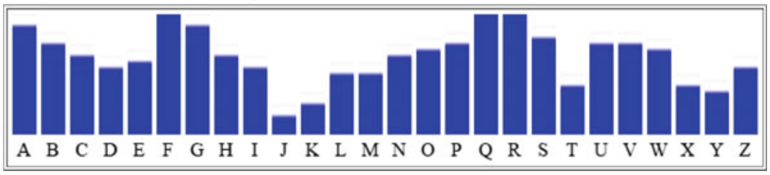


Here is the same plaintext, encrypted with a Vigenère cipher with a key of length 7:

```
ZWXLH XNNME AWEQU OZDHL INAQR HUKAW ECHVV BTFHH
XCEYQ REZQG VQRVO CZSAF PQNYS NOXZP RGIIW PABIF
HHZWY SEAUM FRGOR LJGQP QLREV USBOL VVBTF HLAIN
AQIVB QTSFA WWVNZ AFIQX RHTOX ACARB OLVVF WZSHD
GEOXH DVFFS MCKPC ARUSD ZCPSF OSCVQ CINHF CPHXY
VMXRB TUQLT RRYIO MUOSR OUMAB IDOSX QASZT GWGFP
QYRVF GVQPR QPGGS ALVGQ PKPXB VVBSD REPZW XEVBQ
ASETK MTROD EVMXR FMLRB JRFIA BAQSA MKLVI CCUNW
AVUSS APMKF DXABM FJWFH DLGPY AFRVG UIZDU MFBBQ
CDZFF RUSWI PPSOA ULUES BRHAG AHMNX UDRFA FPQNR
```

```
GFRDD GYSPW KMPBB QIVXN NMQDL BCQRE TKIVZ OZYPQ
NRGFO WPGCZ MYHZU GFUPV WHNFF HLAJN BPHDH CEROA
ULUNF QUVMF GCBRH DGAHK OXZQC DANHV VSFAM SCVGW
ZGGWY ARUSW IPPSO AULUG VQYFI PBBXY EMRYO KEGQH
LCGRR XRBBQ NWPCF OSOFI TQCZT RXQSH TEEIV GZQPL
TGGVQ CDZFF ODERC VBTSA VIEPW PEQBH YOFTL ZGFDQ
EGTKZ WFAQL UGCBR HUGQM OAULU SWJPU WDYSY SFIWF
SPBBP CMODD FITQG BLDGG QCZYR CDLMA UUWRC CZEQB
VUSOA ULUNF QGDAQ YWZEU MRNWD SVXCE SFIUM GARAF
OQOVH MNGOQ FOREW GENFP SSZGI SZTBW WECBP RVGAH
RRRUR HHFIQ OUCSO IIQEU OLAUL ENFPS RVABI UNWPG
SWDSW XNNQQ TKMAN FQEAB TNHMN NLTVJ UNJIE RDGNF
BWESB RRWHN BPRLO JGCRW DGCAR FHHZG NFQOQ TABBQ
OIMCP VUNWP GQSOK
```

And here is the histogram:



Clearly, the letters are much more evenly distributed in the case of the Vigenère cipher than in the case of the substitution cipher. It is typically easy to tell from looking at a histogram which type of cipher it is, if you have to choose between substitution and Vigenère.

We can calculate the index of coincidence for the Vigenère cipher: we find that $I_C \approx 0.0419$. The k value, according to our formula, is then $k \approx 7.89$. Since the key length is 7, that wasn't too bad (although, of course, if we start with the assumption that the key length is 8, we will make no progress).

Instead, looking at letters differing by 7 places only, we get an index of coincidence of around 0.0639; for 14, we get 0.0657; for 21, we get 0.063. These are all close to the value for English, which should convince us that a key length of 7 is likely.

5.6 Problems

(1) Encrypt the plaintext message

```
many cheerful facts about the
square of the hypotenuse
```

using a Vigenère cipher with key `MATH`.

(2) The message

```
JKVSE OZGCF RHOPS LVWKI HZJFV VMLNH FMHSE
MNUOO IUTBV PABOA ULMGN BMZYK JEBNS NIBFJ
JHGSH SIRHI MEEEF WTZYK WCZWW UPBWE QNNBP
PGHIN SDXMC IHZJT GTHSI UPTZX EDZZG USAIJ
IWPBN BVPFX NBWXN NNBQZ QXAZC XPHIL CYXAZ
QFNEI DBOWX HRVOQ MGZXZ LGXDB WDJTD BVTQL
JZFJX HJEBQ SKOPS CIRJC QLRMN PSHCH PZGHI
XONON IBIBV PWMMM SEFNO IPFPE DAHSI OZZMQ
MKNBA LRMCI HJSNH MSERH RQKPR MOWGL MGOXO
FPLOE ODNNN BOQXX MUMWE GYQBR EZMMO ELHPA
SELXT DSMYB GBHSE MCIGD GTMKS CSHHB CDXTI
LWYEG YBVPV XBZOX EVCZS PAHIB MZYMC QBVMM
VRCVI MCMZZ AXMQK SMLKM FOXAZ TCFHX MQGAS
DZBVP RBRMB EXHOP SESPZ ZHZWX ZBVPA BGLPP
ELOAH SMGFQ BRSNO WTXCP DBGES UZNFT KAOMB
OEMGM ODXUP BHSIL ZEWWH UZIGE WBAWI YHLOI
BOMGB BOXIH IIGSI EAVCE SGZWT ELXFQ HSEEA
ACHME YIGXC LZTTY IQOQA LHXAW FELXW IBVWB
MNCCX AZZST ATNBC WHPZZ SZGXV VGZJL DTJPV
TILAZ YGOIW YWHAO CWHUP BWDSH INCFR WOPWD
XTGSK LWFZZ SMPNN BSCEG YDOAS NMNCC XAZOC
WHTIL HSILD TJPVP ZZSLP EHIRP SYKID PVTAZ
WPRWO WCVQX DVHZX AZXOC PBVUS YXAJC GPEGY
BVPVX NIHEL XNXSL OXMIG XYFVA OXSNN MTZVB
IADTX XJNVT WGVUS HSGOG CFXAD VYELB NIXZO
XOPCE LXNXS LOXMP SHLHH BVPCT GTCQX AZUGA
SDZBC ZJTGT HSILO ZOYKX KTONI LDMJP VPVAW
YATNV HELTO VCHXA ZXZLG XAWFL LNWJI MEGYL
WYAAD TSDSF ZUOOI TWWHS IKOWY PIIJB VPVLL
CWPXT ILHSI KZAHN EEGLT ZVHML SCQXV VWYKC
PAHXE DZIFT SMOPS YWAJC ZOCHP PSCIT ABSCF
XOWZO SYNWA PNHFM PJXAZ MBRPB NPAPR FVLSR
EBIAH JSNMW KYGHP VHCCY JTYEI EGBVT WMVTS
XCWZI FSSGZ GOYHL OWIEP RKZCE ILOBV LXHAI
ZWXAZ JIWPF VSSCW CJPBM YEGQG ELXWM GE
```

is a ciphertext encrypted using a Vigenère cipher. What is the plaintext message? What is the key?

(3) The message

```
OBVSE GVAMG TKGWS LTFSV MUDFP ETIHF VYEYM
LBDFL KGWMG NTGJM FKMEU IHYFU LGQAF UUONM
```

```
LYCDS OMDYU FNFZG NVIMG HWFZL HVJMF ETWYL
WHRBI ILLMZ YSWTS FCSII LLCCQ NCXML VIJQO
WMUJV KWAHX FNFED FYNSY MDYJF RGQWN NNTJW
ECCJS KXKBR WDVLW LVXNQ HAMXZ IUMFA FMDQR
LNYJD CCKMV JMNEF ERSDN SFANM EPEDF XTEUV
AAYYA PHFIK MIPKY IVXWT SFARS DKWFN PTUTP
AZVJX VVWGV QYHPS NNNTJ RGNYN NIAZU KJVGV
LIXWU OFDYR YWJIF AVWMC RTUEI SHVGG DZSKG
KNRSD UTDUP JDYEY HVWIO TWLWJ CVJDS ZGAFI
LBVRG QXZYP IIFRL MRDNQ FMNFK FVLWS NJNVH
ALVHT NCLBV TRIEF VFDSV LWSJY ORTWX KMEKV
FIZXE YMLBI JAFMF YJXSW VHLZX IPKSH ULRKR
FCELH GVVMF KHWVV SXZRF WJYKN RGHSJ FQOIM
RCELI QJXYI JDISD XZSSW QKOEY ONHLI RQLYL
GXTTN VVSXZ HTOIA MRNDK HHUPF PQYFX RIAAX
GUEDO PIOBF PIEOW XDJID VAVVI WKXZN FDSIV
WUKAU NKSLS TYUXG OKYET WGGVY HKRYM GNTGJ
MFSZT DPWMJ DOWRG NYJYY SMFUG EUSUI EKOWR
VYUQY RSDCK JFWPA HJMOT XLBVX ECKYL RAAVM
FACFD UXZYP YIEOD YDDTC WLYJY HGCXY VIMAJ
SXJYH GCYCM JMGXZ CJFNF XZYPL IXIEY KMAVE
FXZAE PSLBZ SGYLS NVAET XGAIZ MDPWU K
```

is a ciphertext encrypted using either a substitution cipher or a Vigenère cipher. Which one was used? (You can decrypt it if you would like.)

(4) (a) How many substitution ciphers are there?
  (b) How many substitution ciphers are there with at least one letter fixed (for example, v gets encrypted as V)?
  (c) Exactly one letter fixed?
  (d) Exactly two letters fixed?
  (e) Same questions, but for Vigenère ciphers with key length d. (Part of the challenge here is to figure out exactly what the question is.)

(5) Compare the number of substitution ciphers with the number of Vigenère ciphers with key length 8. Why are Vigenère ciphers with key length 8 harder to break than substitution ciphers?

(6) You now know how to break substitution and Vigenère ciphers. If you had to send a message using one of these ciphers, how would you design your message so as to make decryption as difficult as possible for an attacker? (Note that you can present the same *content* with multiple different messages, by changing the wording.)

Chapter 6
The Hill Cipher

6.1 Matrices

The next cipher we will look at, known as the *Hill cipher*, is based on matrices. It is a form of a substitution cipher, except that it doesn't just substitute one *letter* for another but rather one *block* of letters for another. For example, it might swap some three-letter block with another three-letter block.

Let us start with an overview of matrices. We will just state the facts that we need, without proof, since it would take too long to give all the proofs.

Definition 6.1 Let m and n be positive integers. An $m \times n$ *matrix* is an m by n array of numbers, put into a box. Here, m denotes the number of *rows*, and n denotes the number of *columns*.

Example

$$\begin{pmatrix} 1 & 4 \\ 0 & -7 \\ 19 & 8\pi^3 \end{pmatrix}$$

is an example of a 3×2 matrix.

Typically, we use either round brackets, as above, or square brackets to delimit a matrix. Generally, we fill our matrices with numbers, for example, integers or real numbers. But we may also fill them with other things, as long as we are able to add and multiply those things. In particular, we might want to fill them with elements of $\mathbb{Z}/m\mathbb{Z}$, for some m. And that is what we will need to do for the Hill cipher: our key will be a matrix whose elements lie in $\mathbb{Z}/26\mathbb{Z}$. We will write them as numbers from 0 to 25, but we must recall that their addition and multiplication must be taken modulo 26.

Given two matrices, there is *sometimes* a way of multiplying them to get a new matrix. In particular, we can multiply an $m \times n$ matrix by an $n \times p$ matrix, and the result will be an $m \times p$ matrix. However, it is *not* possible to multiply an $m \times n$

matrix by an $n' \times p$ matrix if $n \neq n'$. The matrix multiplication operation is a bit strange-looking, but it turns out to be a good idea for reasons that we will see in Chapter 19.

Notation Given a matrix A, we often write a_{ij} for the entry in the ith row and jth column.

Definition 6.2 Let A be an $m \times n$ matrix and B an $n \times p$ matrix. Then, we define AB to be an $m \times p$ matrix whose entry in the ith row and jth column is $\sum_{k=1}^{n} a_{ik}b_{kj}$.

Example

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} = \begin{pmatrix} 1 \times 5 + 2 \times 7 & 1 \times 6 + 2 \times 8 \\ 3 \times 5 + 4 \times 7 & 3 \times 6 + 4 \times 8 \end{pmatrix} = \begin{pmatrix} 19 & 22 \\ 43 & 50 \end{pmatrix}.$$

If we assume that these matrices have entries in $\mathbb{Z}/26\mathbb{Z}$, then we would get $\begin{pmatrix} 19 & 22 \\ 17 & 24 \end{pmatrix}$.

Note that sometimes we can form the product AB, but *not* the product BA the other way around. And even when we can, these two products might not be equal.

Example Let

$$A = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}, \qquad B = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}.$$

Then

$$AB = \begin{pmatrix} 1 & 1 \\ 1 & 2 \end{pmatrix}, \qquad BA = \begin{pmatrix} 2 & 1 \\ 1 & 1 \end{pmatrix}.$$

One particularly important special case of matrix multiplication is *matrix-vector multiplication*. If we have a matrix with only one column, we call that matrix a *vector*, or perhaps a *column vector*. (By contrast, there are also *row vectors*, which have only one row. But we will not need those at the moment.) If we have an $m \times n$ matrix, then we can multiply it by a (column) vector of length n, and then result will be a vector of length m.

Example

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix} \begin{pmatrix} 2 \\ 5 \end{pmatrix} = \begin{pmatrix} 12 \\ 26 \\ 40 \end{pmatrix}.$$

There is a special matrix that does nothing when we multiply it by another matrix. This is the *identity matrix*. In fact, there is an identity matrix for each n: an $n \times n$ square matrix whose *diagonal* entries a_{ii} are 1, and everything else is 0. Here is the 4×4 identity matrix:

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

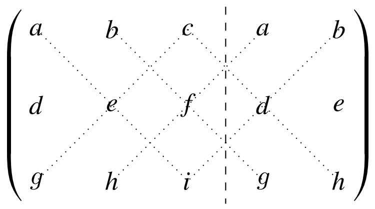
If we let I_n denote the $n \times n$ identity matrix, and A is any $m \times n$ matrix, then $AI_n = A$, and if B is any $n \times m$ matrix, then $I_n B = B$.

Given an $n \times n$ square matrix A, we might wonder if there is another $n \times n$ matrix B so that $AB = I_n$ and $BA = I_n$; this is the matrix version of a *unit*, as discussed in Definition 4.10. When such a matrix B exists, we say that A is *invertible*. We usually write A^{-1} for the matrix B, and it turns out that this inverse matrix, when it exists, is unique.

It turns out that there is a way to check if a matrix is invertible without actually finding the inverse matrix B, just as we saw that $a \in \mathbb{Z}/m\mathbb{Z}$ is a unit if and only if $\gcd(a, m) = 1$, without having to find the inverse b. This relies on the *determinant*. The determinant is slightly complicated to define in general, but we can write it down in the case of a 2×2 or 3×3 matrix. It is only defined for square matrices.

Definition 6.3 If $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$ is an 2×2 matrix, then its determinant is defined to be $\det(A) = ad - bc$. If $B = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix}$ is a 3×3 matrix, then its determinant is defined to be $\det(B) = aei + bfg + cdh - ceg - afh - bdi$.

The 4×4 determinant already contains 24 terms, so the formula is cumbersome, and there are often better ways of computing it. There is a picture that helps in remembering the formula for a 3×3 determinant:



Just add up the products that go from northwest to southeast, and subtract the products that go from northeast to southwest.

It turns out (see, for instance, [Bre12, Theorem 6.2.4] for the case of matrices with entries in $\mathbb{R}$, or [DF04, Theorem 30, §11.4] for the general version) that a matrix A is invertible if and only if its determinant is a unit. Thus, if the entries are in $\mathbb{R}$, it is invertible if and only if its determinant is nonzero. If the entries are integers and we want an inverse that also has integer entries, then this happens if and only if its determinant is ± 1. If the entries are in $\mathbb{Z}/m\mathbb{Z}$, then it has an inverse if and only if $\gcd(\det(A), m) = 1$.

Although the above statement is nonconstructive in that it doesn't tell us how to find the inverse matrix A^{-1}, there is a way of writing down A^{-1} explicitly. We will only do so in the 2×2 and 3×3 cases.

Proposition 6.4 • *If* $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$ *is an invertible* 2×2 *matrix, then its inverse is*

$$A^{-1} = \frac{1}{\det(A)} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix}.$$

• *If* $A = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix}$ *is an invertible* 3×3 *matrix, then its inverse is*

$$A^{-1} = \frac{1}{\det(A)} \begin{pmatrix} ei - fh & ch - bi & bf - ce \\ fg - di & ai - cg & cd - af \\ dh - eg & bg - ah & ae - bd \end{pmatrix}.$$

Here $\frac{1}{\det(A)}$ *means the reciprocal if our matrix entries are in* $\mathbb{Z}$ *or* $\mathbb{R}$ *or something similar, and it's the inverse in* $\mathbb{Z}/m\mathbb{Z}$ *if our matrix entries are in* $\mathbb{Z}/m\mathbb{Z}$*.*

This can be proved, in a rather unenlightening manner, simply by multiplying A by its claimed inverse and checking that the result is in fact the desired identity matrix. Based on Proposition 6.4, we can see why we need the determinant to be a unit: the formula for the inverse involves dividing by the determinant. Note that the formula for matrix inversion is a bit unwieldy to use by hand, but Sage is happy to help us out: if we type

```
A = matrix(Integers(26),[[0,1,2],[3,4,6],[8,1,17]])
A^-1
```

then we get the result

```
[22 19  6]
[ 9 22  8]
[ 9  2  9]
```

And we can check that, modulo 26, it is indeed true that

$$\begin{pmatrix} 0 & 1 & 2 \\ 3 & 4 & 6 \\ 8 & 1 & 17 \end{pmatrix}^{-1} = \begin{pmatrix} 22 & 19 & 6 \\ 9 & 22 & 8 \\ 9 & 2 & 9 \end{pmatrix}.$$

6.2 Encrypting with Matrices

The idea of the Hill cipher is to encrypt blocks of characters using matrices, replacing a block of some small number of letters with a different block of the same size. For instance, suppose we choose to encrypt blocks of length 3 by blocks of length 3. (The plaintext and ciphertext blocks must be the same size for the Hill cipher to work.) To do this, we choose some invertible 3×3 matrix A with coefficients in $\mathbb{Z}/26\mathbb{Z}$ as our key. A common way of doing this is to start with a 9-letter word and then convert it into numbers. We will use the key `TANGERINE`. (I had to try a bunch of different 9-letter fruits before I was able to find one that was usable as a key.) To turn this into a key, write the letters of `TANGERINE` in a 3×3 matrix, as

$$\begin{pmatrix} \mathrm{T} & \mathrm{A} & \mathrm{N} \\ \mathrm{G} & \mathrm{E} & \mathrm{R} \\ \mathrm{I} & \mathrm{N} & \mathrm{E} \end{pmatrix}.$$

Next, convert each letter to a number, by replacing `A` with 0, `B` with 1, `C` with 2, and so forth, up to `Z` with 25. We get

$$\begin{pmatrix} 19 & 0 & 13 \\ 6 & 4 & 17 \\ 8 & 13 & 4 \end{pmatrix}.$$

Its determinant is 5, which is a unit modulo 26, so we're safe. (The reason I had to try a bunch of possible words for keys is that when I tried several other fruit names, such as `PERSIMMON` and `RASPBERRY`, I ended up with a determinant that is not a unit, and hence isn't usable.)

To encrypt a message, we must first convert the plaintext into numbers, using the same scheme: replacing `A` with 0, `B` with 1, and so forth. Let us suppose that we wish to encrypt the plaintext message `rhinoceros`. In order for our encryption scheme to work, we need the message length to be a multiple of the block size, which is 3. Since our message length is 10, we need to append a few random characters at the end so that the message length becomes the next multiple of 3, namely, 12. Once the message is decrypted on the other side, it will be obvious that the extra characters are simply junk and should be discarded. So, let us append `jc` at the end, making our plaintext message `rhinocerosjc`.

We now replace each letter with a number, giving us

```
17 7 8   13 14 2   4 17 14   18 9 2.
```

To form the ciphertext, we turn each block into a column vector of length 3 and multiply it by our key matrix. For instance, with the first block, we get

$$\begin{pmatrix} 19 & 0 & 13 \\ 6 & 4 & 17 \\ 8 & 13 & 4 \end{pmatrix} \begin{pmatrix} 17 \\ 7 \\ 8 \end{pmatrix} = \begin{pmatrix} 11 \\ 6 \\ 25 \end{pmatrix}.$$

Converting that back to letters, we get `LGZ`. Similarly, the other blocks encrypt to `NMI`, `YSX`, and `EWJ`. Putting all these blocks together and rearranging them into our standard length-5 blocks, we get our complete ciphertext:

```
LGZNM IYSXE WJ.
```

Knowing the key and the block length, decryption is very similar to encryption, but instead of multiplying each length-3 block by the key matrix, we multiply it by the *inverse* of the key matrix. In this case, the inverse of the key matrix is $\begin{pmatrix} 11 & 13 & 0 \\ 12 & 10 & 3 \\ 4 & 13 & 10 \end{pmatrix}$. Multiplying this by the vector $\begin{pmatrix} 11 \\ 6 \\ 25 \end{pmatrix}$ corresponding to the first block `LGZ` gives us back the first block of the plaintext, which is $\begin{pmatrix} 17 \\ 7 \\ 8 \end{pmatrix}$, or `rhi`, just as expected.

6.3 Attacking the Hill Cipher

Unfortunately, the Hill cipher is quite weak. Intuitively, this makes sense: it is defined in a very structured way, based on the operation of matrix multiplication. A common theme in cryptography is that the more mathematical structure our cryptosystem has, the weaker it tends to be, as an eavesdropper can use that structure to attack the system.

Let us first suppose that our eavesdropper has managed to acquire a ciphertext encrypted by a Hill cipher, together with a couple extra pieces of information. In particular, she knows that the block size is 2, and she knows that the block `th` encrypts to `BP`, and `at` encrypts to `YL`. It turns out that this alone is enough to calculate the entire key, and thus decrypt the ciphertext.

How can that be? Let us suppose that the 2×2 key matrix is $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$. Based on the two plaintext–ciphertext pairs, Eve knows that

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} 19 \\ 7 \end{pmatrix} = \begin{pmatrix} 1 \\ 15 \end{pmatrix}, \quad \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} 0 \\ 19 \end{pmatrix} = \begin{pmatrix} 24 \\ 11 \end{pmatrix}. \tag{6.1}$$

Writing that out in terms of equations, we get the following system of four linear equations in four variables:

$$\begin{aligned} 19a + 7b &= 1 \\ 19c + 7d &= 15 \\ 0a + 19b &= 24 \\ 0c + 19d &= 11. \end{aligned}$$

(And remember, all these equations are modulo 26.) So, to find a, b, c, d, we simply solve the system of equations. In fact, this system is easier than a usual system of four equations in four variables, because it splits up into two separate systems of equations, each of which consists of two equations in two variables. When we solve it, we find that

$$a = 15, \qquad b = 4, \qquad c = 0, \qquad d = 17.$$

Converting this back to letters, we see that the key is `PEAR`.

We can rephrase this process in matrix form. The pair of equation (6.1) is equivalent to the single matrix identity

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} 19 & 0 \\ 7 & 19 \end{pmatrix} = \begin{pmatrix} 1 & 24 \\ 15 & 11 \end{pmatrix}.$$

We wish to solve for a, b, c, d, which means inverting the matrix $\begin{pmatrix} 19 & 0 \\ 7 & 19 \end{pmatrix}$. Its inverse is $\begin{pmatrix} 11 & 0 \\ 11 & 11 \end{pmatrix}$, which Sage is happy to tell us with the following commands:

```
plaintext = matrix(Integers(26),[[19,0],[7,19]])
plaintext^-1
```

Now that we have the inverse, we can solve for $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$:

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} = \begin{pmatrix} 1 & 24 \\ 15 & 11 \end{pmatrix} \begin{pmatrix} 11 & 0 \\ 11 & 11 \end{pmatrix} = \begin{pmatrix} 15 & 4 \\ 0 & 17 \end{pmatrix}.$$

This is the same answer as before.

More generally, if Eve knows that the block size is n, and she can somehow acquire n different plaintext–ciphertext blocks, then she can determine the key and use it to decrypt any ciphertext message. But how is Eve to get these plaintext–ciphertext blocks? Ordinarily, Alice and Bob will not be eager to provide them to Eve. So, one possibility for Eve is to get them from the ciphertext itself, if it is long enough. She may be able to do this using frequency analysis. If she decrypts a very long message and knows that the block size is 2, then it is very likely that the most common block is `th` in the plaintext, giving her one block. She can try various other common bigrams like `he`, `in`, and `er` for other common bigrams in the ciphertext. With a bit of time and patience, she should be able to find her necessary two bigrams to decrypt the ciphertext.

There is a bit of a pitfall here though: it's not enough just to have *any n* different plaintext–ciphertext blocks. Rather, they have to be sufficiently special in order for this attack to be completely successful. In the above example, the plaintext matrix $\begin{pmatrix} 19 & 0 \\ 7 & 19 \end{pmatrix}$ is *invertible*: it has an inverse, considered as a matrix with entries in $\mathbb{Z}/26\mathbb{Z}$, because its determinant is relatively prime to 26. This won't always happen: the determinant could be divisible by 2 or 13. When that happens, these plaintext–ciphertext blocks won't be sufficient to work out the key directly, but it can lead to partial information about the key. We encourage you to try problem 3 to see how to get this partial information, and to determine how it might be used.

6.4 Problems

(1) Encrypt the plaintext message

```
Why, sometimes I've believed as many as
six impossible things before breakfast!
```

with a Hill cipher using the key PORCUPINE.

(2) The message

```
VCKUP EDOPS JICJP NBZCV DDMKI IRQKP WAKQI
QMJEX HSQAH XHSZX LCTC
```

is a ciphertext encrypted by a Hill cipher with block size 3. Through a breach in security, you determine that the plaintext nectarine is encrypted as RCSXLBCFC. Determine the key and the plaintext of the longer ciphertext.

(3) The message

```
ZWZII MBBSQ KMKNN QKYKM ZZWZU ALXPN QKYKM
ZZWZI IMBBS QKMAA A
```

is a ciphertext encrypted by a Hill cipher with block size 3. Through a breach in security, you determine that the plaintext andbutfor is encrypted as AHWZTFLRZ. Determine the key and the plaintext of the longer ciphertext. (Hint: Gur xrl vf n sbezre anzr bs n pbhagel.)

Chapter 7
Other Types of Ciphers

7.1 The Autokey Cipher

The autokey cipher, suggested by Vigenère himself as an improvement on the cipher that now carries his name, is similar to the Vigenère cipher in that it is polyalphabetic: it uses different shift amounts for different letters. However, it is stronger, in that it cannot be attacked in the same way as the Vigenère cipher can. Let us see how it works.

The setup is as before: we have a key and a plaintext. We start by using the key to encrypt the plaintext, just as with the Vigenère cipher, but once we have used up the entire key, instead of repeating it, we use the beginning of the plaintext to encrypt the rest of the message.

Example Suppose our plaintext is `thecatinthehatisback` and our key is `CODE`. Then we encrypt as follows, breaking everything into groups of five as usual:

Plaintext	`theca`	`tinth`	`ehati`	`sback`
Key	`CODET`	`HECAT`	`INTHE`	`HATIS`
Ciphertext	`VVHGT`	`AMPTA`	`MUTAM`	`ZBTKC`

Hence, our ciphertext is `VVHGT AMPTA MUTAM ZBTKC`.

An attack like the Kasiski examination won't work on the autokey cipher, since there is no reason to believe that a string of letters *ever* gets encoded in the same way twice. However, there is another attack available to Eve. The idea is to guess a word that appears in the plaintext and use that to string along the rest of the message. Let us see this attack in action.

As we mentioned earlier, the most common trigram in English is `the`, so Eve might think it likely that combination will appear somewhere in the message. Well, if `the` appears somewhere in the message, then unless it's very close to the end, it will show up in the key somewhere. Now, Eve doesn't know the position it will appear in the key, so she has to try all possibilities. However, she can try them all

S. Rubinstein-Salzedo, *Cryptography,* Springer Undergraduate Mathematics Series, https://doi.org/10.1007/978-3-319-94818-8_7

fairly quickly, because she can begin by supposing that the first letter will appear in some spot congruent to 1 (mod 3), then 2 (mod 3), then 0 (mod 3). In the case of 1 (mod 3), she makes her supposed key `THETHETHE...` and uses this key to decrypt the ciphertext. When she does this, she obtains the following:

```
Ciphertext  VVH GTA MPT AMU TAM ZBT KC
Key         THE THE THE THE THE THE ..
Plaintext   cod nmw tip hfq ati gup ..
```

She will then similarly look at the 2 (mod 3) and 0 (mod 3) cases. Here is the 2 (mod 3) case:

```
Ciphertext  V  VHG  TAM  PTA  MUT  AMZ  BTK   C
Key         .  THE  THE  THE  THE  THE  THE   .
Plaintext   .  cac  ati  wmw  tnp  hfv  img   .
```

And here is the 0 (mod 3) case:

```
Ciphertext  VV HGT  AMP  TAM  UTA  MZB  TKC
Key         ..  THE  THE  THE  THE  THE  THE
Plaintext   ..  ozp  hfl  ati  bmw  tsx  ady
```

Now, what Eve does is to look at all the trigrams that are possible in the plaintext. Some of these seem like plausible sequences in the plaintext, and some of them don't. The most likely ones are probably `cod` and `ati` (the latter repeated three times in fact, in different places). So, she will guess that one of these is actually right. She doesn't know which one, so she has to try all of them, or at least enough of them so that she gets one to work.

Eve doesn't know the length of the key, so she has to guess a few possibilities. They key probably won't be under three characters, so she might start with a key of length 3. Suppose she tries the `ati` starting at position 13, with a key of length 5. What she will do is to use this bit of the key to generate more of the key and plaintext. Here is what will happen: she originally has

```
Ciphertext  VVHGT AMPTA  MUTAM ZBTKC
Key         ..........  ..THE .....
Plaintext   ..........  ..ati .....
```

Now, if the key has length 5, then `the` appears in the plaintext five places before it appears in the key, as does `ati`. Thus, she obtains

```
Ciphertext  VVHGT AMPTA MUTAM ZBTKC
Key         ...... ..WMW ..THE ..ATI
Plaintext   ...... ..the ..ati ..tru
```

We can keep stringing the plaintext and key along in this way. However, this one is probably wrong, since `WMW` is not a likely piece of the key (and hence the plaintext). Now, Eve has lots of free time to experiment with all the likely possibilities, so she will mess around with various key lengths with `ati` before moving on. But we get bored more easily, so let's jump to the right answer, which is `ati` with a key length of 4. We then obtain the following after one step:

Ciphertext	`VVHGT`	`AMPTA`	`MUTAM`	`ZBTKC`
Key	`.....`	`...AT`	`I.THE`	`.ATI.`
Plaintext	`.....`	`...th`	`e.ati`	`.bac.`

See how much better this looks: all the trigrams look like plausible combinations of English letters. Continuing on, we obtain:

Ciphertext	`VVHGT`	`AMPTA`	`MUTAM`	`ZBTKC`
Key	`COD.T`	`HE.AT`	`I.THE`	`.ATI.`
Plaintext	`the.a`	`ti.th`	`e.ati`	`.bac.`

We now know the entire message except for every fourth character. But any guess for the fourth letter (or the eighth, or the twelfth, etc.) will feed through and give us all the rest of the characters, so Eve can try all 26 possibilities and see which one gives her something that makes sense. If she chooses `C` for the fourth letter of the message, she will recover the original message `the cat in the hat is back`.

In practice, it is very difficult to break such a short autokey cipher. As we have seen, this is a general trend: longer messages are easier to break than shorter ones, because they have more structure to them.

7.2 Transposition Ciphers

The ciphers we have seen so far replace letters "in place," but they never swap letters around. However, there are common ciphers that involving moving ciphers around. We will now discuss two of them: the *rail fence cipher* and the *columnar transposition cipher*. They are both easy to describe.

In the rail fence cipher, we begin by writing our message as though moving down and up diagonally on the rails of a fence. The key consists of a positive integer, for the number of rows on the fence. For instance, suppose our plaintext is `thekeyisunderthemat`, and the key is 4. Then, we write the message like this:

```
T.....I.....R.....T
.H...Y.S...E.T...A.
..E.E...U.D...H.M..
...K.....N.....E...
```

To form the ciphertext, we simply read along the rows. After breaking the message into blocks of length 5 as usual, we get a ciphertext of `TIRTH YSETA EEUDH MKNE`.

Given the key, it is not too difficult to recover the plaintext from the ciphertext. Let's have a look, using the ciphertext we just got.

Our message has length 19, so we need to start with a 4×19 grid of dots:

```
...................
...................
...................
...................
```

Based on the construction of the rail fence cipher, we know which dots get filled, so let's mark those spots with a different symbol, say `x`:

```
x.....x.....x.....x
.x...x.x...x.x...x.
..x.x...x.x...x.x..
...x.....x.....x...
```

So we just fill those spots in order from top to bottom from the ciphertext:

```
T.....I.....R.....T
.H...Y.S...E.T...A.
..E.E...U.D...H.M..
...K.....N.....E...
```

Finally, to recover the plaintext, just read diagonally.

The rail fence cipher isn't very good for security, because there aren't very many different plausible keys; for instance, it doesn't make much sense if the key length is longer than the plaintext length, so even for 1000-character messages, there are only 1000 possible keys. That number can easily be checked by computer, or even somewhat tediously by hand.

The other transposition cipher we will look at is the *columnar transposition cipher*. For this one, we write our message in some number of columns. Let us use the same plaintext as before and use five columns. Our message then looks like this:

```
THEKE
YISUN
DERTH
EMAT.
```

Next, we choose some permutation of the columns. A common way of doing this is to choose a word whose length is equal to the number of columns, preferably with no repeated letters. Let us use the key MANGO. Of the letters in the word MANGO, the alphabetically first letter is the A, then the G, and so forth. Its *pattern* is 31425: the first letter of the word is alphabetically third (hence, the 3 at the beginning), the second letter of the word is alphabetically second (hence, the 1 in the second digit), and so on. Let us write the pattern as the top row of the message:

```
31425
THEKE
YISUN
DERTH
EMAT.
```

To get the ciphertext, just read down the columns, starting from the one labeled 1, then the one labeled 2, and so on. Putting them into blocks of length 5 as usual, we get the ciphertext HIEMK UTTTY DEESR AENH.

Given the key, this process is easily reversible to allow recovery of the plaintext from the ciphertext and key.

Like the rail fence cipher, the columnar transposition cipher is quite weak. However, there is a possibility that we have not considered yet: using a double-layer cipher. That is, we can *first* apply a transposition cipher, and *then* apply a second cipher, like a substitution cipher or Vigenère cipher.

Is a double-layer cipher good security? That depends on the ease of peeling off one of the layers of the cipher. The combination of a transposition cipher and a substitution cipher is somewhat better than a substitution cipher alone, but is not wholly satisfactory. Recall that we attack a substitution cipher through frequency analysis. Single-letter frequencies do not change after we have scrambled around the letters via a transposition cipher, so that bit of the attack still works. But the rest of the attack, through multi-letter frequencies, fails and cannot be used against this combination. So, for a long message, it *might* be possible to make progress against the substitution portion alone. The combination of transposition and Vigenère, on the other hand, seems stronger: the Kasiski examination falls apart completely, and while we can still gain insight into the key *length* from the Friedman coincidence index, getting at the actual key seems difficult.

7.3 One-Time Pads

So far, all of the ciphers we have seen (Cæsar, substitution, Vigenère, Hill, autokey, and transposition) can be attacked based on peculiarities in their construction. There are other ciphers that are much, much harder or impossible to attack. A good example is the one-time pad. The idea of a one-time pad is the same as that of a Vigenère cipher or an autokey cipher, except instead of repeating the key or using the message as the key, we just choose a very long key; in other words, a one-time pad is a Vigenère cipher in which the key is the same length as the message. A good way of obtaining a long key is simply to have a book of random letters and use them as the key. Provided that Alice and Bob both have the same book of random letters, and an agreement about where to start in the book, they can easily send messages that the other one can decipher, and Eve has no hope of breaking the code unless she can somehow obtain the book.

It is important that Alice and Bob only use each line (or whatever) of the book once, as Eve's chances to break it will increase if they use it repeatedly. Reusing a one-time pad is a serious breach of security and can result in the code being compromised.

The problem for Eve in trying to decipher a code made using a one-time pad is that, if she intercepts a message, she gets no information about its content except its length. For example, she might intercept the message `JMABE`, and she might guess that it means `today`. And indeed, there is some key that would encrypt `today` as `JMABE`. But there is also some key that would encrypt `later` as `JMABE`, and she has no way of distinguishing these two possibilities. In fact, one-time pads are *provably* unbreakable: given a ciphertext encrypted with a one-time pad, it is equally likely to be an encryption of *any* plaintext message whose length is the same as that of the ciphertext. Amazingly, the one-time pad is the only cryptosystem which is proven to be completely secure; there are others, such as the ones we will look at later in the book, that we *believe* to be secure, but we have no proof yet, and there are currently major obstacles in the way before we can hope to prove any such thing.

One-time pads (actually printed on silk) were used by British spies in World War II. The Germans could only break codes by capturing the spies and stealing their one-time pads. Spies were frequently captured, so the British needed a way of detecting that this had happened. They arranged to vary from what was printed on the one-time pad slightly, in a way that was different for each spy. The Germans knew this, but they didn't know how to guess what the modification was. A description of some of the British cryptographic efforts, especially the spies' use of one-time pads and the previously used poem codes, in World War II can be found in [Mar01].

The problem with one-time pads is that they are very cumbersome: you have to use a different one-time pad for every message you send, so before any messages are sent, both sides have to make sure that they both have all the pads, and that they are kept in order. If they plan to send messages repeatedly, that makes for a lot of pads! In short, one-time pads are good in theory, but are annoying to use in practice.

7.4 Introduction to Modern Cryptosystems

We have now seen a number of alphabetical ciphers and analyzed their vulnerabilities. All of them have their problems that we have already discussed. However, they also have one more very serious problem that we have not discussed at all yet. The issue is that Alice and Bob must agree on a key and an encryption scheme before they send any messages. Now, if Alice and Bob are friends and see each other in person and Eve doesn't get to attend those meetings, then this is fine. But what if Alice and Bob have never met each other? How do they agree on a key?

This is the setup that is of great importance in the modern world: suppose Bob owns an internet store, and Alice wants to buy something from his store. Naturally, she has to tell Bob her credit card number. But she does not want Eve to know what it is. How can Alice and Bob agree on a key that allows her to send her credit card number without Eve being able to figure out what the key is?

This sounds like an impossible task: anything that they send to each other can be intercepted, but somehow the messages have to be intelligible to each other but not to Eve. It is truly remarkable that this can actually be done. Before we look at the details of such a system, let us explain the philosophy behind it: Alice and Bob don't have to give Eve a problem that's *impossible* to solve; it's good enough if they give her a problem that takes 100 000 years to solve. Eve has a lot of time, but even she doesn't have *that* much time.

How can they do that? The point is that there are problems in mathematics that are **easy to solve but hard to unsolve**. Or, at least, they appear to be hard to unsolve; we don't have proofs that they are hard to unsolve. However, they have resisted attempts by many great mathematicians, so we are quite confident that, at least in practice, they are hard to unsolve.

We mentioned earlier the problem of factoring numbers when we discussed the gcd: in the naïve algorithm for computing the gcd, we had to factor the numbers, and that takes a while, even on a computer. But of course we have a better algorithm for computing the gcd. At any rate, factoring numbers appears to be a problem that is hard to solve. However, factoring numbers is the opposite of a different problem: multiplying numbers. So, this is the first example of a problem that is easy to solve and hard to unsolve: if I give you two large numbers, you can multiply them together easily. However, if I give you one large number and tell you it's the product of two primes, it is very hard to find those primes! We will look for similar problems in order to create secure modern cryptosystems.

In order to motivate the construction of modern cryptosystems, we'll begin with a little puzzle whose solution contains some rudiment of the ideas of modern cryptography:

Puzzle 7.1 *Pretend that you cannot trust the government (or maybe you don't have to pretend). You need to send a box, with something valuable inside, to a friend in a distant city. You expect the people at the post office to open the box and steal the contents, unless you lock the box, which can accept one or more locks. Luckily, padlocks with keys are common. If you lock the box with your padlock, the people* *at the post office will not bother to try to pick the lock; there are easier crimes to commit elsewhere. But if you use a padlock, you will have to send the key to your friend, and the key will be stolen en route. Even a letter with a photograph of the key will be stolen. Even if you can fax a picture of the key, the people at the phone company will also receive your fax and send it on to the post office. In other words, you cannot send a copy of the key to your friend. How can you send the box, so that only your friend can open it?*

This seems hard: there is no way to send a lock that your friend can open. Fortunately, however, you don't have to do so! Here is how you can do it:

First, send your box with a padlock on it. Your friend can't open it, but ey[1] can put a *second* padlock on the box and then return it to you. You then remove your lock and send it back, with only your friend's lock remaining. Your friend can then remove eir padlock and open the box.

This puzzle is an introduction to the idea of *shared information*: if Alice and Bob can somehow develop a mechanism to share some information (a key) that Eve can't understand, then they can send each other unbreakable messages. If we abstract this puzzle a bit to its core, we see that the moral is that, even though Eve theoretically has everything she needs to read the message, in practice she cannot do it. This is a big theme in cryptography.

With this in mind, we can now *abstractly* describe the notion of an encryption scheme. An encryption scheme has several pieces:

- A set K of possible keys.
- A set P of possible plaintext message.
- A set C of possible ciphertext messages.
- A function $e : K \times P \to C$ that takes in as its inputs a key and a plaintext and uses them to produce a ciphertext.

In this formalism, we only require the encryption scheme to encrypt messages, not to decrypt them as well. However, being able to encrypt messages without also being able to decrypt them isn't especially valuable, so we also need the corresponding notion of a *decryption scheme*. This consists of the following pieces:

- A set K' of possible keys.
- A set C of possible ciphertext messages.
- A set P of possible plaintext messages.
- A function $d : K' \times C \to P$ that takes a key and a ciphertext and produces out of it a plaintext message.

In general, encryption and decryption schemes ought to be paired with each other: given a plaintext message, if we first encrypt it and then decrypt it, we ought to get the original message back. So, we might want the following to be true:

[1]This is an example of a *Spivak pronoun*, a third-person singular gender-neutral pronoun. To form them, remove the "th" from the third-person plural pronouns, with the exception that "themselves" becomes "emself." I'm on a mission to get these pronouns into common English parlance. Please do your part to support the cause.

$$d(k, e(k, p)) = p$$

for all plaintext messages $p \in P$ and for all keys $k \in K$.

However, there is a small subtlety here, that we have hinted at in the notation: we potentially allow ourselves to have *two* sets of keys: K and K'. Here, K is the set of encryption keys, and K' is the set of decryption keys. In all the encryption and decryption schemes we have seen so far, we use the same keys for encryption and decryption, but that doesn't have to be the case, and indeed it won't be true in the modern cryptosystems we study later in the book. That doesn't mean that we can encrypt with any key whatsoever and decrypt with any key and hope to recover the original message; instead the decryption key has to be related to the encryption key somehow. Thus, we require that if k and k' are "corresponding" encryption and decryption keys, respectively, then

$$d(k', e(k, p)) = p.$$

As we will see, it is generally easy to start with a decryption key k' and produce the corresponding encryption key k, but the reverse is difficult. Once again, think about multiplication versus factoring: k' might consist of two prime numbers p and q, whereas k might be their product pq. Certainly k contains the information of k' in theory, but how are we supposed to access it in a reasonable amount of time and effort?

A *cryptosystem* is a corresponding pair of an encryption scheme and a decryption scheme. It is useful to distinguish two types of cryptosystems: symmetric and asymmetric cryptosystems. In a symmetric cryptosystem, the encryption key and the decryption key are the same. This is the case in all the classical ciphers we have studied so far: if you have the information needed to encrypt a plaintext message, you can also decrypt a ciphertext easily using the same key. On the other hand, in an asymmetric cryptosystem, the encryption key and the decryption key are different (but related). Modern cryptography, as it is implemented today, relies on both symmetric and asymmetric cryptosystems. The cryptosystems we will look at in this book will all be asymmetric ones. This isn't because they are more important in practice; serious cryptosystems in place today use a combination of symmetric and asymmetric cryptosystems. However, we choose to focus on the asymmetric cryptosystems, because there is no question that they are far more interesting from a mathematical point of view. As this is first and foremost a book about mathematics, we favor topics that are mathematically exciting over others of equal real-world significance.

We'll need a fair amount of mathematics before we can define these modern asymmetric cryptosystems, so we'll spend some time doing some number theory and abstract algebra now, as well as analysis of algorithms.

7.5 Problems

(1) Encrypt the plaintext message

```
To ask you what you mean to do
we punctually appear
```

using an autokey cipher with key `EULER`.

(2) Encrypt the plaintext message

```
Brevity is the soul of wit
```

using a rail fence cipher with five rails.

(3) Encrypt the plaintext message

```
Better three hours too soon than a
minute too late
```

using a transposition cipher with key `LOQUAT`.

(4) The message

```
BTATE TYNFI TIFAO WTWOH OHEAO TLSRL I
```

is a ciphertext been encrypted using a rail fence cipher. What is the plaintext message? What is the key?

(5) The message

```
XVZKA LATSF HHKQR BBWOE ZBIVI YFASP ARALQ
MTBJM MNMPC BOGAL RLOES ULGTP HXIGM LAWFG
YLWHF FYLSL ARWLM XFHDX IERAH XCRCT AIYKX
HTXWO XYGHX XLRRY HPOID MCNAW FGAAG RRWQR
IHWLL VXVWW PXQEI PIGRR WQRIH WLLVW PSFLE
OONUG OYDVQ XOYOD QXHYC WIEOP JTKAP WCACL
IELPA AIOSS IVQXO YODQX HYCWI EOTON NGCMR
RBVBI KYVVJ ERBMM USRUJ MKLAQ KVXTA PUSEC
APJTK APRJF OUWER UIPCC GTILS NUQXG JULGV
SXKCP OELZM ZMTBJ WBLTO KSGTZ KXLRO HEGTZ
KTEKV NSYIX YGSFA SCCIN WRVIS PRHCC GLVQC
QTVVI LKULV CVMWS SXRAD OSPRG RHCCG MFHMR
LKVDR RRGBR HCCGV LXNPS HCTGT ZKXLR OHEGT
ZKXLR OHEGT ZKWNC GAIGH CCBPO UPWTZ YAJDB
HRLPL AXGPW ESVTL SRDVX LRFLR XIZEM RVXPU
WQTGT GDUFR UQYWY XPWFO FXHRV NUGOY DVQXO
YODQX HYCWI EOPJT KAPWC ACLIE LPAAI OSSIV
QXOYO DQXHY CWIEO TONNG CMRRB VBIKY VVJER
BHQJI TMNRO ZNXBW QXAGO CXGXR REDWG MKXWP
NILSV TJADM GIPAY BTTIE OUATI XPGOT QZMDT
WCIYZ QJSWC WTBIU PAIUE MRIEI ILEMM FVNFB
VTVBN BRQGQ QYJPP HVYNE SPRKQ ENLRE QEZMN
```

```
AQFBX DOWEV PPPWS OMEWS GVEXZ YZBKY YWZTK
AHBVS CVIGO FURWO AMWWV JHDWY NZRKH QUSEC
APJTK AHRYY ORIQH GEAIK
```

is a ciphertext that has been encrypted using an autokey cipher. What is the plaintext message? What is the key?

(6) Suppose that we wish to do a double-layer encryption of a plaintext, encrypting it with a substitution cipher and a transposition cipher. Show that it does not matter which order we do the encryption. Show that it *does* matter if we use a Vigenère cipher and a transposition cipher.

(7) There are 45 coins on a table, ten showing heads and the rest tails. They are covered with a cloth. You must reach under the cloth and separate them into two groups, not necessarily equal in number, turning over any and as many coins as you like. You cannot tell in any way (by feeling or some other method) whether a coin is showing heads or tails. When you are finished and the cloth is removed, both groups should show an equal number of heads. How do you do it?

Chapter 8
Big O Notation and Algorithm Efficiency

8.1 Big O Notation

When we do a computation, we would like to do it as fast as possible, or at least pretty fast. In order to talk about (roughly) what that means, we need to introduce big O notation.

Definition 8.1 Let $f(x)$ and $g(x)$ be two functions, we say that $f(x) = O(g(x))$ if there is some constant $C > 0$, which does not depend on x, so that

$$|f(x)| \le Cg(x)$$

for all x.

Remark 8.2 There is a slight subtlety here: what does *for all* x mean? It could mean for all $x \in \mathbb{R}$, but we will use it slightly more weakly to mean for all *sufficiently large* x, i.e., there is some A so that $|f(x)| \le Cg(x)$ for all $x \ge A$.

The point of big O notation is that it allows us to focus on the most important features of the function and not the less important ones.

Example If $f(x) = x^3 + 5x^2 - 2x + 131$, then $f(x) = O(x^3)$. More generally, if $f(x)$ is a polynomial of the form ax^n+ lower order terms, then $f(x) = O(x^n)$. We also have that $f(x) = O(x^{n+1})$ or $O(x^{n+2})$, but typically we will be interested in finding a big O that's close to the best we can do.

There are various functions that we like to use for $g(x)$ when using big O notation. We are fond of functions that combine polynomials, logarithms, and exponentials, especially. So, $O(x^2)$, $O(x^3)$, $O(\log(x))$, $O(2^x)$ are commonly used.

Another notation we will use is $O(x^{2+\varepsilon})$, or something like that. In mathematics, we generally use ε to refer to a very small positive number. So $f(x) = O(x^{2+\varepsilon})$ means that for any $r > 2$, $f(x) = O(x^r)$, but it might not be the case that $f(x) = O(x^2)$. Generally, this means that $f(x)$ is roughly x^2 times something with logarithms in it.

Let us remind ourselves (or see for the first time) of what logarithms are and what they're plotting.

8.2 Logarithms

Definition 8.3 Let $b > 0$, $b \neq 1$. If $c > 0$, we define $\log_b(c)$ to be the unique real number a so that $b^a = c$.

Example • $\log_3(81) = 4$ since $81 = 3^4$.
• $\log_2\left(\frac{1}{2}\right) = -1$, since $\frac{1}{2} = 2^{-1}$.
• $\log_4(2) = \frac{1}{2}$ since $4^{1/2} = 2$.

It is important to know not just the definition of logarithm, but also its properties. Here are some of the most important ones:

- $\log_b(1) = 0$.
- $\log_b(xy) = \log_b(x) + \log_b(y)$.
- $\log_b(x^a) = a\log_b(x)$.
- $\log_b(a) = \frac{\log(a)}{\log(b)}$, where log denotes a logarithm to any base of your choosing.
- $\log_b\left(\frac{1}{a}\right) = -\log_b(a)$.

All these properties follow easily from the definition.

We generally assume that the base b of the logarithm satisfies $b > 1$, to make it behave as we expect it to. Your favorite value of b will determine the sort of person you are:

- If you are a mathematician, your favorite value of b will be $e \approx 2.718281828\ldots$ The base-e logarithm is sometimes called the "natural logarithm" since, well, it's the most natural for doing mathematics. Beginners sometimes write ln for the natural logarithm because we still pretend that we're speaking Latin or something. But in fact, it's so natural that mathematicians usually just call it the logarithm without any modifiers.
- If you are a computer scientist, your favorite value of b will be 2, because everything on a computer is done in binary. The base-2 logarithm tells you (more or less) the number of binary digits (bits) required to represent a number.
- If you are not yet a mathematician or computer scientist, then your favorite value of b will be 10, because you think of numbers as being written in base 10. When you finish reading this book, I hope you will be a mathematician, so that you can choose a sensible base of logarithm, rather than determining it based on the historical accident of the number of fingers that most humans have. (Also for many other reasons!)

We will usually just write log to denote a logarithm to an arbitrary base $b > 1$, but since I'm a mathematician, I will implicitly think of the base as being e.

For many purposes, the base of the logarithm is not too important. This is, especially, true where algorithms and big O notation are concerned: for any $b, c > 1$, $\log_b(x) = O(\log_c(x))$, and vice versa. To see this, note that $\log_b(x) = \frac{\log_c(x)}{\log_c(b)}$, so $\log_b(x)$ and $\log_c(x)$ differ only by a multiplicative constant, which is exactly the sort of thing that big O notation measures.

Another important thing to know about the logarithm function is its growth rate in terms of big O notation.

Exercise Show that, for any $\varepsilon > 0$, $\log(x) = O(x^{\varepsilon})$.

Thus, while $\log(x) \to \infty$ as $x \to \infty$, it gets there very slowly. More slowly than any positive power of x, for that matter. But there are functions that go to ∞ *even more slowly*. For example, $\log(\log(x))$, and further iterates of $\log(x)$. And in fact, there are even more slow-growing functions: iterates of logarithm go to ∞ very rapidly when compared to the *inverse Ackermann function*, which is beautifully described by Raimund Seidel in [Sei]. Anyway, we won't need the inverse Ackermann function at any point in this book.

A typical function that is $O(x^{2+\varepsilon})$ would be

$$x^2(\log(x))^3\sqrt{\log(\log(x))}.$$

8.3 Running Times of Algorithms

We will frequently use big O notation when referring to the running time of algorithms: how long does it take to run a program to do something? Now, there is a slightly subtle point to address before we proceed: when we think of performing some task, we might think of giving the program some number n and having it output another number N, and we might be tempted to ask how long it takes to compute N in terms of n. This is almost right, but it doesn't generalize that well.

Suppose that instead of having a program that takes in a number and outputs another number, we have a program that takes in some string of characters and outputs another string of characters. How do we measure the size of the input now? The natural answer is that we measure the *length* of the input string. (Another option would be to convert the input string into a number, but this is a bit silly, and there isn't necessarily a preferred way of doing this.)

So, if we measure the running time of an algorithm involving an input *string* in terms of the length of the string, perhaps we should do that for an algorithm involving an input *number*: we measure it in terms of the number of *digits* in the number. This is (up to a constant) $\log(n)$. Furthermore, the running time in big O terms that we report will generally measure the *worst-case* running time: the longest it can possibly take for an input of a given length. (There are other possibilities, such as the average case running time, that are also important.)

So, now let's look at some simple algorithms and work out their running times.

Example (*Addition*) Given two numbers with at most n digits, how long does it take to add them? In order to add them, we have to add the 1's digits, which takes a constant number of steps (i.e., it doesn't depend on the lengths of the numbers). Then, we must add the 10's digits, plus a possible carry, which also takes a constant number of steps. And so forth. Hence, it takes a constant number of steps for each digit, and there are n digits, so it takes about n steps. The running time of this algorithm is $O(n)$.

Example (*Multiplication*) To multiply two numbers with at most n digits, we basically need to multiply every pair of digits. Every digital multiplication takes a constant number of steps, and there are n^2 multiplications we need to do, so this takes n^2 steps. Then, we have to add up all our results, which takes another n^2 or so steps. So, multiplication runs in $O(n^2) + O(n^2) = O(n^2)$ steps.

In fact, this method is not optimal, at least for multiplying very large numbers. Faster algorithms include Karatsuba's algorithm [Kar95], which runs in $O(n^{\log_2(3)}) \approx O(n^{1.585})$ steps, the Toom–Cook or Toom-3 algorithm [Coo66], which runs in $O(n^{\log_3(5)}) \approx O(n^{1.465})$ steps, and the Schönhage-Strassen algorithm [SS71], which runs in $O(n \log(n) \log\log(n))$ steps. While these have faster *asymptotic* running times, they are slower for multiplying relatively small numbers.

We now describe Karatsuba's algorithm. (We develop the Toom–Cook algorithm, which is similar, in problem 3.) The idea is as follows: Suppose we wish to compute 38×67. Ordinarily, we would multiply each digit of 38 by each digit of 67, and then do a bunch of additions. In short, we have to do four single-digit multiplications and a bunch of additions. We let

$$\begin{aligned} z_2 &= 3 \times 6, \\ z_1 &= 3 \times 7 + 8 \times 6, \\ z_0 &= 8 \times 7, \end{aligned}$$

so that $38 \times 67 = z_2 \times 100 + z_1 \times 10 + z_0$. (Note that multiplication by 10 or 100 doesn't really require a multiplication, since it's just shifting over by one or two digits.)

Can we do it with only three multiplications? Yes: $z_1 = (3+8) \times (6+7) - z_2 - z_0$, and we have already computed z_2 and z_0, so we don't have to recompute them. Okay, we aren't quite multiplying together single digits, but we're close enough.

Let us now investigate how to multiply large numbers in general. Suppose we wish to multiply two n-digit numbers x and y. The idea is to divide x and y into two $n/2$-digit numbers. (If n is odd, do something with the rounding; it doesn't really matter. We'll just assume n is even, to avoid messy floors and ceilings.) So, we write $x = x_H \times 10^{n/2} + x_L$ and $y = y_H \times 10^{n/2} + y_L$. Then

$$\begin{aligned} xy &= (x_H \times 10^{n/2} + x_L)(y_H \times 10^{n/2} + y_L) \\ &= x_H y_H \times 10^n + (x_H y_L + x_L y_H) \times 10^{n/2} + x_L y_L. \end{aligned}$$

It seems that we need to perform multiplications to compute $x_H y_H$, $x_H y_L$, $x_L y_H$, and $x_L y_L$. But in fact, we can do it faster: we compute

$$\begin{aligned} a &= x_H y_H \\ d &= x_L y_L \\ e &= (x_H + x_L)(y_H + y_L) - a - d. \end{aligned}$$

Then, we have

$$xy = a \times 10^n + e \times 10^{n/2} + d.$$

What is the running time? Suppose $T(n)$ is the number of steps needed to compute the product of two n-digit numbers. We converted this problem into three $n/2$-digit multiplication problems, each of which takes $T(n/2)$ steps (when done using Karatsuba's algorithm, if $n/2$ is bigger than 1, say, and otherwise just done the usual way), plus a few extra "maintenance" steps. How many maintenance steps? In computing e, we had to add the two $n/2$-digit numbers x_H and x_L, and similarly y_H and y_L. By our previous analysis of addition, these each take $O(n/2) = O(n)$ steps. Similarly, subtracting a and d each takes $O(n)$ steps. Thus, it required a total of $O(n) + O(n) = O(n)$ maintenance steps. Thus, we have found that, using Karatsuba's algorithm, the number of steps for multiplication is given by the recurrence

$$T(n) = 3T(n/2) + O(n).$$

So, we iterate:

$$\begin{aligned} T(n) &= 3T\left(\frac{n}{2}\right) + O\left(\frac{3}{2}n\right) \\ &= 9T\left(\frac{n}{4}\right) + O\left(\frac{9}{4}n\right) \\ &= \cdots \\ &= 3^k T\left(\frac{n}{2^k}\right) + O\left(\left(\frac{3}{2}\right)^k n\right). \end{aligned}$$

Choosing $k = \log_2(n)$, we have

$$T(n) = O\left(3^{\log_2(n)}\right) + O\left(\left(\frac{3}{2}\right)^{\log_2(n)} n\right) = O\left(n^{\log_2(3)}\right).$$

An important takeaway message from Karatsuba's algorithm and the other fast multiplication methods is that sometimes the most obvious algorithm for solving a problem is not the fastest. Sometimes, it pays to look for something cleverer.

Example (*Sorting*) Suppose we have a list of numbers or strings in some random order, and we want to put them in alphabetical or numerical order. Let us suppose that we have n items in our list. How long does it take to sort? This is actually quite a hard question, and there are better and worse algorithms for sorting. The stupidest sorting algorithm, but also the easiest, is called BUBBLE SORT. The way it works is as follows: we start by comparing the first two items in the list. If they're in the right order, keep them the way they are. If they're in the wrong order, swap them. Once that is done, move on to the second and third items in the list. If they're right, keep them, and if they're wrong, swap them. Continue on with the third and fourth, and so on.

Once we have finished that pass, start again, redoing the whole process. Keep going until there are no pairs in the wrong order. One can show that it takes at most $n - 1$ passes for the list to be sorted.

So, what is the running time? We have to do $O(n)$ passes, and we have to do $O(n)$ steps in each pass, so it takes $O(n^2)$ steps to sort the list.

There are better algorithms, which sort a list in $O(n \log(n))$ steps. This is a substantial improvement, so BUBBLE SORT is never used in practice for large lists. One of these fast sorting algorithms is called MERGE SORT. We aim to sort a list of n elements. For convenience, we will assume that $n = 2^m$ is a power of 2; some small adjustments are needed when this is not the case. To do this, divide the list into two smaller lists, each with $n/2 = 2^{m-1}$ elements, sort each one, and merge the results.

This seems a little circular at first: we have to sort the smaller lists, but how do we do that? Exactly the same way as before: we break each of the lists of length 2^{m-1} into two smaller lists, each of length 2^{m-2}, and keep breaking them down in this way. We stop when we get to a list of length 1, since any list of length 1 is automatically sorted.

How do we merge two lists? It is easy to imagine doing this with two piles of index cards, each of which contains a number. Take the first (smallest) card from each pile and see which one is smaller. The smaller of the two becomes the smallest in the merged pile. Now, take the smallest remaining card in each of the two piles and compare those. Keep going in this manner until all the cards are used. If each pile is of size k, then it takes (at most) $2k$ comparisons to merge the two piles.

Now, how long does it take? Let us suppose that it takes $f(n)$ steps to sort a list of length n, assuming that $n = 2^m$ is a power of 2. We break up the pile into two smaller piles, each of size $n/2$, and then we have to do three things: sort the first pile, sort the second pile, and merge the two piles. Each of the two sorts requires $f(n/2)$ steps, and the merge requires (at most) n steps. Hence, we have

$$f(n) \leq 2f\left(\frac{n}{2}\right) + n$$

steps. It is not clear what this function looks like yet, so instead we build it up, starting from $f(1) = 0$. Using the recurrence, we find that

$$\begin{aligned} f(1) &= 0 = 1 \times 0, \\ f(2) &\le 2 \times 0 + 2 = 2 = 2 \times 1, \\ f(4) &\le 2 \times 2 + 4 = 8 = 4 \times 2, \\ f(8) &\le 2 \times 8 + 8 = 24 = 8 \times 3, \\ f(16) &\le 2 \times 24 + 16 = 64 = 16 \times 4, \end{aligned}$$

and so on. In general,

$$f(n) = f(2^m) \le m2^m = n\log_2(n),$$

so $f(n) = O(n\log(n))$.

There are many other clever sorting algorithms, which you can see with animations [Sor].

Example (*Searching*) Suppose we have a list of n items that is *already* sorted, and we want to find an entry in it. (You might not be old enough to remember doing this with telephone books, but you can imagine trying to find someone's phone number by looking it up in an alphabetized directory.) How long does this take?

Again, it depends how you do it. You could do it by starting on the first page and going through every single entry in the book until you've found it. If there are n entries in the phone book, it takes $O(n)$ steps to do this. But this is really stupid. You should instead start by trying to get close and then refine the search.

Well, it's hard to program a computer to try to get close, or to say what that means precisely, but we can at least improve upon the naïve algorithm by starting in the middle and seeing if we're too high or too low. If we're too high, we eliminate the second half of the list and go halfway in the first half. We then keep on narrowing it down in this manner, at each step eliminating half the list from the previous step. This takes $O(\log(n))$ steps.

Example (*Primality testing*) Given a number N, how can we tell whether it is prime? One way of doing this is to divide it by all the numbers less than N and see if it is exactly divisible by any of them. Or, we can be a little bit smarter: if N factors as $N = ab$, then either a or b must be at most $\sqrt{N}$, so we really only have to divide by numbers up to $\sqrt{N}$. This involves $O(\sqrt{N})$ steps. However, recall that N only has $n = O(\log(N))$ digits, so $N = O(10^n)$. Hence, this algorithm for primality testing runs in $O(\sqrt{10^n})$ steps. Except not quite, because each "step" is a division, which itself requires something like $O(n^2)$ steps. Thus, overall, this algorithm for primality testing runs in $O(\sqrt{10^n} \times n^2)$ steps.

However, there are much faster algorithms for primality testing. The best provable algorithm is due to Agrawal, Kayal, and Saxena (AKS algorithm), which in its original form runs in $O(n^{12+\varepsilon})$ steps, and more recent modifications have brought it down to $O(n^{6+\varepsilon})$ steps. There are other algorithms that seem to do better, but that are hard to prove. For example, the Miller–Rabin algorithm, which is a *probabilistic* algorithm (i.e., one that relies on choosing random numbers and thus doesn't work every single time but that usually does) that can be turned into a deterministic algorithm if we assume that the Generalized Riemann Hypothesis is true, runs in time $O(n^{4+\varepsilon})$ steps. The elliptic curve primality test seems to run in $O(n^{4+\varepsilon})$ steps, but we are not able to prove this. We will return to the topic of primality testing later in Chapter 13, but for now, we simply refer the interested reader to [Pom10].

Once again, we see that sometimes the number of steps it takes to do something depends on the algorithm we use to do it.

The point of all of this is that we need to know how long it will take to run some sort of algorithm. Let's return to the topic of cryptography. Alice and Bob have to work out a key between them. And it can't take too long for them to do so. If it takes 1 000 000 steps, that's probably okay to do on a computer because computers are fast, but if it takes 10^{30} steps, that isn't okay, because computers aren't *that* fast.

In general, we think of algorithms that run in polynomial time in the length of the input $O(n^k)$ for some k (or better) as being fast, and algorithms that run in exponential time $O(a^n)$ for some a as being slow. This is not necessarily completely true: an algorithm that runs in time $O(n^{100})$ is probably too slow to be practical. However, such algorithms don't arise very frequently; usually an algorithm that runs in polynomial time runs in time $O(n^5)$ or better.

In the setting of cryptography, since Alice and Bob have to create a key from information that they pass back and forth between them, Eve certainly has all the information available to her to break it. So, what Alice and Bob have to do is to arrange for it to take many steps for her to break it. If it suddenly takes her 10^{30} steps to break it, then she can't do it in practice, even if she could "theoretically" in the sense of having all the information she needs to specify it uniquely. What we'll be looking for is problems that take a very long time to solve. In order to make such problems, we need some more number theory.

8.4 Problems

(1) (a) Let $f(x) = x^3 + 5x + 7$. Is $f(x) = O(x^2)$? Why or why not?
    (b) Let $f(x) = x^4 + 6$. Is $f(x) = O(x^5)$?
    (c) Let $f(x) = x^6 + 992x^3 - 541$. Is $f(x) = O(e^x)$?

(2) Find algorithms and running times for solving the following problems:

    (a) Check if a string says "Hello."
    (b) Check if a number contains the digit 4.
    (c) Check if a string is a palindrome.
    (d) Find the longest palindrome formed by taking consecutive letters in a string. (For example, if your string is"ACDEFGGFEDH," then the longest palindrome is "DEFGGFED.")
    (e) Check if a list of numbers contains any number more than once.
    (f) Check if a list of numbers is sorted.

(3) When we multiply two n-digit numbers x and y using Karatsuba's algorithm, we split them each into two $n/2$-digit parts. Could we do better by splitting them into some different number of parts? In terms of big O notation, how many steps does multiplication take if we divide them into k parts? Feel free to assume that n is a power of k in your analysis. (Hint: Show that when $k = 3$, you can multiply x and y by performing 5 multiplications of $n/3$-digit numbers. Generalize.) This is the Toom–Cook Algorithm. Ordinary long multiplication is Toom-1. Karatsuba's algorithm is Toom-2. Toom-3 is asymptotically faster.

Chapter 9
Abstract Algebra

9.1 Some Group Theory

Much of what we will do later on in cryptography will be phrased in the language of group theory, so it will be a good idea to get used to it now. Group theory is a bit abstract, but we'll try to keep it as hands-on as possible. We begin by defining the very general notion of a group:

Definition 9.1 A *group* is a set G together with an operation $\cdot$ (which we think of as being multiplication) taking two elements g and h of G and giving back a third element $g \cdot h$ of G, with the following properties:

(1) The operation is associative: $a \cdot (b \cdot c) = (a \cdot b) \cdot c$.
(2) There is an identity element, which we call e. This element has the property that, for any $g \in G$,
$$e \cdot g = g \cdot e = g.$$
(3) Every element has an inverse. That is, for every element $g \in G$, there is some element $g^{-1} \in G$ so that
$$g \cdot g^{-1} = g^{-1} \cdot g = e.$$

Remark 9.2 A group need not be commutative (also called abelian): that is, $a \cdot b$ need not be equal to $b \cdot a$. Noncommutative (also known as non-abelian) groups are extremely important in mathematics, but we will not need them to do cryptography. So, all the groups we see in this book will be abelian.

Remark 9.3 When the group operation is multiplication, we will frequently suppress the $\cdot$ and write ab in place of $a \cdot b$.

Remark 9.4 Abelian groups are named after the nineteenth-century Norwegian mathematician Niels Henrik Abel. True fame in mathematics is having your name turned into an improper adjective.

We are already familiar with many examples of groups.

S. Rubinstein-Salzedo, *Cryptography,* Springer Undergraduate Mathematics Series, https://doi.org/10.1007/978-3-319-94818-8_9

Example The integers $\mathbb{Z}$ under addition form a group. The identity element is 0, and the inverse of $n \in \mathbb{Z}$ is $-n$.

Remark 9.5 When dealing with abelian groups, we frequently (but not always!) write $+$ for the group operation (rather than $\cdot$) and call the identity element 0.

Example The integers $\mathbb{Z}/m\mathbb{Z}$ modulo m under addition also form a group. Some people write this group as $\mathbb{Z}_m$, but this is wrong, and you should never do that.

Example $\mathbb{Z}/m\mathbb{Z}$ under multiplication does not form a group, but the units $(\mathbb{Z}/m\mathbb{Z})^\times$ do form a group. Observe that if a and b are both units, then ab is also a unit. To see this, suppose that a^{-1} and b^{-1} are the inverses of a and b, respectively. Then, $a^{-1}b^{-1}$ is the inverse of ab.

It will be useful for us to *find* the inverse of some number $a \in (\mathbb{Z}/m\mathbb{Z})^\times$. We have already seen how to do this, but let us recall how to do it: we need to find some $b \in (\mathbb{Z}/m\mathbb{Z})^\times$ so that $ab \equiv 1 \pmod{m}$. This amounts to finding b, c so that

$$ab + mc = 1.$$

We do this using the Euclidean algorithm.

There is a special type of group that will be especially important for us. This is a group in which every element is some power of one fixed element.

Definition 9.6 A group G is said to be *cyclic* if there is some $g \in G$ such that, for every $h \in G$, $h = g^n$ for some integer n.

Example For any m, $\mathbb{Z}/m\mathbb{Z}$ is a cyclic group. The group of integers $\mathbb{Z}$ is also cyclic.

Any cyclic group is abelian, because powers of a single element commute. However, there are abelian groups which are not cyclic. Problem 6 at the end of the chapter is to find one.

9.2 Fields

We will be especially interested in $\mathbb{Z}/m\mathbb{Z}$ when m is a prime number p. In this case, $\mathbb{Z}/p\mathbb{Z}$ has more structure than that of a group. Under addition, $\mathbb{Z}/p\mathbb{Z}$ is a group, but we also have multiplication. We know that the elements of $\mathbb{Z}/p\mathbb{Z}$ that are relatively prime to p form a group under multiplication, but that's just all the elements except for 0. When a set has two such group structures, a group under addition and a group under multiplication, we call the set a field. More precisely:

Definition 9.7 A *field* is a set F, together with two operations $+$ and $\cdot$ so that F forms an abelian group under addition, and the nonzero (that is, nonidentity) elements of F also form an abelian group under multiplication, and the distributive property is satisfied:

$$a(b + c) = ab + ac.$$

In short, a field is a place where we can add, subtract, multiply, and divide (by things that aren't zero).

When we think of $\mathbb{Z}/p\mathbb{Z}$ as a field, rather than just a group under addition, we write $\mathbb{F}_p$, which stands for the field with p elements.

Exercise If m is composite, why is $\mathbb{Z}/m\mathbb{Z}$ not a field?

We are used to other examples of fields: the rational numbers $\mathbb{Q}$ and the real numbers $\mathbb{R}$ are also fields. However, the integers $\mathbb{Z}$ do not form a field, since there are nonzero integers which do not have multiplicative inverses. (For example, the inverse of 6 would be $1/6$, but this is not an integer!)

9.3 Problems

(1) Which of the following are groups? For the ones that are groups, determine the identity, and describe a method for finding inverses.

- (a) $\{\pm 1\}$ under multiplication.
- (b) The integers under the operation $m * n = -m - n$.
- (c) $\{1, 2, 4\}$ under multiplication modulo 7.
- (d) $\{1, 4, 5\}$ under multiplication modulo 9.
- (e) $\{0, 4, 8\}$ under addition modulo 12.

(2) Show that, when m is composite, $\mathbb{Z}/m\mathbb{Z}$ under addition and multiplication does *not* form a field.

(3) If G is a group, and $g \in G$ is an element, then the *order* of g is the least positive integer n such that $g^n = e$. (If no such n exists, then we say that g has infinite order.) Show that if G is a cyclic group with n elements and $m \mid n$, then G has an element of order m. (This is also true for noncyclic groups, but the proof is a bit more difficult. Can you do it? What if we assume that G is abelian?)

(4) We know that, for every prime p, there is a field with p elements, namely $\mathbb{F}_p$.

- (a) Is there a field with four elements?
- (b) Is there a field with six elements?
- (c) Given a number n, how can you tell whether there is a field with n elements?

(5) (a) Show that if p is prime, then the elements of $\mathbb{F}_p$ are exactly the roots of $x^p - x = 0$.

(b) Show more generally that if $\mathbb{F}_q$ is a field with q elements, then the elements of $\mathbb{F}_q$ are exactly the roots of $x^q - x = 0$.

(6) Find an abelian group with finitely many elements that is not cyclic. (Hint: Ybbx ng gur vairegvoyr vagrtref zbqhyb z sbe inevbhf inyhrf bs z.)

Chapter 10
A Second Look at Number Theory

10.1 Fermat's Little Theorem

The following result will be really important for us:

Theorem 10.1 (Fermat's Little Theorem) *If p is a prime number, any a is any integer, then $a^p \equiv a \pmod{p}$.*

This is sometimes also written as follows:

Corollary 10.2 *If p is a prime number and a is relatively prime to p, then $a^{p-1} \equiv 1 \pmod{p}$.*

Note that this is not Fermat's Last Theorem, which is extremely difficult and took over 350 years to solve. This one we can do on our own.

Proof We'll prove the Corollary instead of the Theorem. Let us start by listing the first $p-1$ multiples of a. They are:

$$a, 2a, 3a, \ldots, (p-1)a. \tag{10.1}$$

We claim that all of these numbers are different modulo p. Suppose not, and that two of the above numbers, say ra and sa, are congruent modulo p, i.e., $ra \equiv sa \pmod{p}$. Then

$$(r-s)a \equiv 0 \pmod{p}. \tag{10.2}$$

Since a is not a multiple of p, it has an inverse a^{-1} modulo p; multiply both sides of (10.2) by a^{-1} to obtain $r - s \equiv 0 \pmod{p}$, i.e., $r \equiv s \pmod{p}$. But all the numbers from 1 to $p-1$ are different modulo p. Thus the only way that we can have $ra \equiv sa \pmod{p}$ is if $r = s$. As a result, all the numbers in (10.1) are different modulo p. Since none of them can be $0 \pmod{p}$, they are the numbers $1, 2, 3, \ldots, p-1 \pmod{p}$, in some order.

Now, multiply them all together: we have

$$\begin{aligned}(a)(2a)(3a)\cdots((p-1)a) &\equiv (p-1)!\,a^{p-1}\\ &\equiv 1\times 2\times 3\times\cdots\times(p-1)\\ &\equiv (p-1)! \pmod{p}.\end{aligned}$$

Now, divide the equation

$$(p-1)!\,a^{p-1} \equiv (p-1)! \pmod{p}$$

by $(p-1)!$ to obtain

$$a^{p-1} \equiv 1 \pmod{p},$$

as desired. ■

10.2 Euler's Theorem

There is a generalization of Fermat's Little Theorem to composite numbers, called Euler's Theorem. (This name, unfortunately, is not terribly descriptive, as there are many theorems named after the great 18th-century Swiss mathematician Leonhard Euler.)

Theorem 10.3 (Euler's Theorem) *If n is any positive integer and a is relatively prime to n, then $a^{\phi(n)} \equiv 1 \pmod{n}$.*

Proof The proof is very similar to that of Fermat's Little Theorem. Look at the numbers of the form ka, where $1 \le k \le n$ and $\gcd(k,n) = 1$. Just as before, all these numbers will be different modulo n, so they are exactly the numbers modulo n that are relatively prime to n. Hence, multiplying them all together, we have

$$\prod_{\substack{k=1\\ \gcd(k,n)=1}}^{n} (ka) \equiv a^{\phi(n)} \prod_{\substack{k=1\\ \gcd(k,n)=1}}^{n} k \equiv \prod_{\substack{k=1\\ \gcd(k,n)=1}}^{n} k \pmod{n}.$$

Dividing both sides by the product, we obtain

$$a^{\phi(n)} \equiv 1 \pmod{n}.$$ ■

10.3 Primitive Roots and the Structure of $\mathbb{F}_p^\times$

When we see that $a^{p-1} \equiv 1 \pmod{p}$, we might wonder why that happens to be the case. Clearly, the powers of a eventually start cycling around; the theorem says that they do that in at most $p-1$ steps; in fact, the length of a cycle *divides* $p-1$.

But could it actually *be* $p-1$? That is, there are $p-1$ possibilities for the powers of a. Might they all be attained?

Clearly, this isn't always the case: for example, if $a \equiv 1 \pmod{p}$, then all the powers of a are also $1 \pmod{p}$, so we'll never hit the rest of them. But, it turns out that sometimes we will hit all of them. More precisely:

Theorem 10.4 (Primitive Root Theorem) *Let p be a prime. Then there is some $g \in \mathbb{F}_p^\times$, the multiplicative group of nonzero elements of $\mathbb{F}_p$, so that all elements of $\mathbb{F}_p^\times$ are powers of g. For such an element g, $g^{p-1} \equiv 1 \pmod{p}$, but $g^r \not\equiv 1 \pmod{p}$ for $1 \le r \le p-2$.*

Typically, one proves this with some group theory and possibly field theory as well. The proof won't be very useful to us though, so we instead refer the reader to [NZM91, §2.8] for a proof.

What this means is that $\mathbb{F}_p^\times$ is a cyclic group. Remember that a group is said to be cyclic, if it consists only of the powers of one element. The groups $\mathbb{Z}/m\mathbb{Z}$ under addition are all cyclic groups, as is $\mathbb{Z}$. (In fact, these are, in some sense, the only cyclic groups, with $\mathbb{F}_p^\times$ being equivalent to $\mathbb{Z}/(p-1)\mathbb{Z}$.)

In fact, we can specify exactly the number of such primitive roots modulo p. Since $\mathbb{F}_p^\times$ is equivalent (or *isomorphic*) to $\mathbb{Z}/(p-1)\mathbb{Z}$, we can count the number of *generators* of $\mathbb{Z}/(p-1)\mathbb{Z}$, or more generally of $\mathbb{Z}/m\mathbb{Z}$. (A generator of $\mathbb{Z}/m\mathbb{Z}$ is an element g so that all elements are of the form ng for some integer n.) Which elements of $\mathbb{Z}/m\mathbb{Z}$ are generators? These are exactly the ones that are relatively prime to m. Hence, there are $\phi(m)$ of them.

Corollary 10.5 *There are $\phi(p-1)$ primitive roots in $\mathbb{F}_p^\times$.*

Unfortunately, while we know how many of them there are, it isn't so easy to tell which elements actually *are* primitive roots. There is no known formula for primitive roots, and indeed we do not even have a simple criterion for determining whether 2 is a primitive root modulo p. This is related to *Artin's primitive root conjecture*, which says that the proportion of primes p such that 2 is a primitive root modulo p is

$$\prod_{p \text{ prime}} \left(1 - \frac{1}{p(p-1)}\right).$$

Similar expressions exist for numbers other than 2. See [Mor12] for a recent survey on the topic.

It will be useful for us not just to know that primitive roots exist, but also to find them. One way of doing so is to guess some number a and check whether $a^r \equiv 1 \pmod{p}$ for some $0 \le r < p-1$. Since there are a lot of primitive roots, if we ever need to find one, we will probably be able to do so in just a few guesses.

Remark 10.6 It might seem strange that, while $\mathbb{F}_p^\times$ and $\mathbb{Z}/(p-1)\mathbb{Z}$ are isomorphic (i.e., have the same structure), it is easy to find a generator for $\mathbb{Z}/(p-1)\mathbb{Z}$ (for example, 1 is a generator) but hard to find one for $\mathbb{F}_p^\times$. This is because the isomorphism between the two groups is abstract: we can prove that one (or in fact many of them) exists, but the proof doesn't tell us what it is. This is quite common in mathematics, and it is usually considered to be something of a disappointment. Here, however, we can be grateful for the difficulty, as it is what makes the Diffie–Hellman cryptosystem (which we shall see in the next chapter) safe. This is a common phenomenon in cryptography: we can turn things that are hard to find in mathematics into strong cryptosystems.

Remark 10.7 We have seen that if a is relatively prime to p, then $a^{p-1} \equiv 1 \pmod{p}$. But if n is composite, might a^{n-1} still be congruent to $1 \pmod{n}$? This doesn't usually happen, but it can. When this happens, we say that n is a pseudoprime (if it happens for some specific choice of a) or a Carmichael number (if it happens for all such a).

10.4 Numbers in Other Bases

Typically, when we write numbers, we write them in base 10. For example, 265 means

$$265 = 2 \times 10^2 + 6 \times 10^1 + 5 \times 10^0. \tag{10.3}$$

However, we can write numbers in other bases just as well. Typically, to write a number in base b, we write 265_7, which means that the number "265" is to be read in base 7. To convert it to base 10, we use (10.3) but replace all the 10's with 7's:

$$265_7 = 2 \times 7^2 + 6 \times 7^1 + 5 \times 7^0 = 145_{10},$$

so the number we write as 265 in base 7 is the same as the number we write as 145 in base 10. In general, in a number in base b, we write it using the digits $0, 1, 2, \dots, b-1$. If $b > 10$, then we typically use some other symbols (such as letters) to denote the digits after 9; for example, in hexadecimal, or base 16, we traditionally use the digits 0–9, together with the letters A–F.

In the last paragraph, we saw how to take a number written in base 7 (or any base, for that matter) and convert it into base 10. We can also go the other way. Let us write 343 in base 5. There are many ways of doing this, but we will just look at one of them. Start by finding the largest power of 5 less than 343; it is $125 = 5^3$. This means that $343_{10} = (abcd)_5$, where $0 \leq a, b, c, d \leq 4$, so that

$$343 = a \times 5^3 + b \times 5^2 + c \times 5^1 + d \times 5^0.$$

We begin by computing a. We divide 343 by 125, which is 2 with a remainder of 93. So, $a = 2$, and we have to deal with the 93 bit next. To compute b, we divide 93 by $5^2 = 25$, which is 3 with a remainder of 18, so $b = 3$. To find c, we divide 18 by $5^1 = 5$, which is 3 with a remainder of 3. Thus $c = 3$ and $d = 3$. Thus we have $343_{10} = 2333_5$.

Aside from base 10, the most important base for us is base 2, also known as binary. In binary, all the digits are either 0's or 1's. Let us do an example with binary.

Example Let us compute 53 in binary. The largest power of 2 less than 53 is $32 = 2^5$, so

$$53_{10} = (abcdef)_2 = a \times 2^5 + b \times 2^4 + c \times 2^3 + d \times 2^2 + e \times 2^1 + f \times 2^0.$$

Since $53 > 32$, $a = 1$. Now, we have $53 - 32 = 21$ left to deal with. Next, we compute b. Since $21 > 16, b = 1$, and we have $21 - 16 = 5$ left. Since $5 < 8, c = 0$. But $5 > 4$, so $d = 1$, and we have to deal with the remaining $5 - 4 = 1$. Now, $1 < 2$, so $e = 0$, and $f = 1$. Thus, $53_{10} = 110101_2$.

10.5 Fast Computation of Powers in $\mathbb{Z}/m\mathbb{Z}$

Another thing we will need to do is to compute a lot of powers in $\mathbb{Z}/m\mathbb{Z}$. That is, we might like to compute $5^{666723489232} \pmod{23425145234}$. How can we do this quickly?

There are two points.

(1) We can decrease the number of steps by repeated squaring. So, if we wanted to compute 5^{100}, instead of multiplying 5 by itself 100 (or 99) times, we could compute the sequence

$$5^1, 5^2 = 5 \times 5, 5^4 = 5^2 \times 5^2, 5^8, 5^{16}, 5^{32}, 5^{64}.$$

Now, squaring again is too big, but we can finish up by noting that $5^{100} = 5^{64} \times 5^{32} \times 5^4$. Thus, instead of taking $n - 1$ multiplications to compute a^n, it takes $O(\log(n))$, which is much better.

(2) When we perform each computation, we don't have to remember the entire answer, but only the remainder modulo m.

Example Let us compute $5^{100} \pmod{33}$. Here is a table of our computations:

$$\begin{aligned}
5^1 &\equiv 5 \\
5^2 &\equiv 5 \times 5 \equiv 25 \\
5^4 &\equiv 25 \times 25 \equiv 31 \\
5^8 &\equiv 31 \times 31 \equiv 4 \\
5^{16} &\equiv 4 \times 4 \equiv 16 \\
5^{32} &\equiv 16 \times 16 \equiv 25 \\
5^{64} &\equiv 25 \times 25 \equiv 31 \\
5^{96} &\equiv 31 \times 25 \equiv 16 \\
5^{100} &\equiv 16 \times 31 \equiv 1.
\end{aligned}$$

In terms of binary, what we did was to write 100 in binary, as 1100100. We computed all the powers of 5 up to 5^{64} by repeated squaring; we stopped at 64 because it is the largest power of 2 less than 100. After that, we multiplied together all the numbers of the form 5^{2^a}, where the a^{th} digit (starting from the right, and counting up from 0) of 100 in binary is a 1. These correspond to $a = 2, 5, 6$, so that $5^{100} = 5^{2^2} \times 5^{2^5} \times 5^{2^6}$. But since we are working modulo 33, we reduced everything modulo 33 whenever we could.

Remark 10.8 It is possible to improve on this repeated squaring algorithm, so that you don't have to store the list of power-of-two exponents, but so that you only have to perform the same number of multiplications as above. Can you see how?

10.6 Multiplicative Functions

Before we move back into cryptography, let us continue our detour into number theory by looking at multiplicative functions. The theory of multiplicative functions will help us to compute $\phi(n)$, among other things. But before we do that, let us work on a slightly easier problem, namely that of computing the number of divisors of n. We will write $d(n)$ for the number of divisors of n. We include 1 and n as being divisors of n here and everywhere else in number theory.

Let us work out the case of $n = 60$, carefully. We could write down all the divisors of 60, but it makes more sense to collect them in a well-organized manner. For example, we can first look at all the ones that aren't divisible by 2. Now, there are few enough of them that it is easy to write them all down: they are 1, 3, 5, and 15. Now, we should look at the ones that *are* divisible by 2, but there is a natural division among those numbers: those that are divisible only by 2, and those that are also divisible by 4. Among those that are only divisible by 2 (that is, not divisible by 4), we have 2, 6, 10, and 30. Note that, each one of these is exactly twice of one of the numbers not divisible by 2. Finally, for those divisible by 4, we have 4, 12, 20, and 60. Thus, there are 12, which are conveniently split up into three groups of four each.

We can apply this sort of organization more generally: if p is some prime dividing n, and p^e is the largest power of p dividing n, then there are the same number of factors of n not divisible by p, divisible by p but not by p^2, and so forth, up to those divisible by p^e. Hence, there are $e + 1$ groups, each of the same size.

In each one of these groups, we could break down still further. Now, since all the groups have the same size, it suffices to focus on one of them, and the natural choice is the set of divisors not divisible by p. Now, we can pick some other prime p' that divides n/p^e and run through the same process.

When we keep breaking it down in this way, we find that, in order to count the divisors of n, it suffices to count the divisors of p^e for each prime p. This last problem is really easy: the divisors of p^e are just smaller powers of p, namely $1, p, p^2, \ldots, p^e$, so $d(p^e) = e + 1$.

To get our final answer for $d(n)$, we simply multiply together all the $d(p^e)$'s for the primes p dividing n. Hence:

Theorem 10.9 *Suppose that the prime factorization of n is*

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

where all the p_i's are distinct primes. Then

$$d(n) = \prod_{i=1}^{r} (e_i + 1).$$

Example Since $60 = 2^2 \times 3 \times 5$, we have

$$d(60) = (2 + 1)(1 + 1)(1 + 1) = 12,$$

which is what we got above as well.

So, this is the easiest way of counting divisors of n. This was easy, because $d(n)$ is an example of a multiplicative function.

Definition 10.10 A function $f : \mathbb{N} \to \mathbb{R}$ (or $\mathbb{C}$, or any other reasonable collection of numbers) is called *multiplicative* if, whenever $\gcd(a, b) = 1$, $f(ab) = f(a)f(b)$.

Note that we do *not* require that $f(ab) = f(a)f(b)$ when $\gcd(a, b) > 1$. Indeed, this fails in the case of $d(n)$ above.

Whenever we have a multiplicative function f, we can work out $f(n)$ as soon as we know $f(p^e)$, for all prime powers p^e. That is, the values of f at prime powers determine f at all positive integers. Thus, to specify f, we might just say what it is at the prime powers.

There are other important multiplicative functions. For example, the sum-of-divisors function $\sigma(n)$ is multiplicative. For $\sigma(n)$, we add up all the divisors of n, instead of just counting them. We can write down a formula for $\sigma(n)$, which is only a little bit more complicated than the formula for $d(n)$.

Theorem 10.11 *If p is a prime, then*

$$\sigma(p^e) = 1 + p + p^2 + \cdots + p^e = \frac{p^{e+1} - 1}{p - 1}.$$

And, of course, the totient function ϕ is also multiplicative. Let us see why. Suppose that $\gcd(m, n) = 1$. We wish to figure out how many numbers there are relatively prime to mn. Write the numbers from 1 to mn in a table, like this:

$$\begin{array}{ccccc}
1 & 2 & 3 & \cdots & m \\
m+1 & m+2 & m+3 & \cdots & 2m \\
2m+1 & 2m+2 & 2m+3 & \cdots & 3m \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
(n-1)m+1 & (n-1)m+2 & (n-1)m+3 & \cdots & nm
\end{array}$$

There are some columns in which there can't be any numbers relatively prime to mn. A typical column contains numbers of the form $km + \ell$, where ℓ is fixed and k can change. By the Euclidean algorithm, $\gcd(km + \ell, m) = \gcd(m, \ell)$, and $\gcd(km + \ell, m) \leq \gcd(km + \ell, mn)$. Hence, if $\gcd(m, \ell) > 1$, then there is no way that $km + \ell$ can be relatively prime to mn.

Hence, there are $\phi(m)$ columns that could conceivably contain numbers relatively prime to mn. In fact, they're all relatively prime to m, so we just have to check how many of them are relatively prime to n. Let us do this by focusing on one such column, say the one consisting of numbers of the form $km + \ell$. I want to show that no two numbers in this column are congruent modulo n. Suppose that $rm + \ell \equiv sm + \ell \pmod{n}$. Then, we can subtract ℓ from both sides to get $rm \equiv sm \pmod{n}$, or $(r - s)m \equiv 0 \pmod{n}$. But since $\gcd(m, n) = 1$, m has an inverse modulo n, so we can multiply by the inverse to get $r \equiv s \pmod{n}$. But $0 \leq r, s \leq n - 1$, so if $r \equiv s \pmod{n}$, they are actually equal. Hence, no two numbers in this column are congruent modulo n, and since there are n numbers in the column, they fill up all the classes modulo n. We know that, of the classes modulo n, exactly $\phi(n)$ are relatively prime to n. Hence, in this column, $\phi(n)$ are relatively prime to n. Since there are $\phi(m)$ relevant columns, we have

$$\phi(mn) = \phi(m)\phi(n),$$

so ϕ is multiplicative.

We have:

Theorem 10.12 *If p is a prime, then*

$$\phi(p^e) = p^e - p^{e-1}.$$

This is because the numbers up to p^e which *aren't* relatively prime to p^e are exactly the multiples of p, and there are p^{e-1} of these.

Example Let us compute $\phi(105)$. We have $105 = 3 \times 5 \times 7$, so

$$\begin{aligned}
\phi(105) &= \phi(3)\phi(5)\phi(7) \\
&= (3-1)(5-1)(7-1) \\
&= 2 \times 4 \times 6 \\
&= 48.
\end{aligned}$$

One way of writing $\phi(n)$ is as follows: suppose n factors as $n = \prod_{i=1}^{r} p_i^{e_i}$, where the p_i's are distinct primes, and each $e_i \geq 1$. Then, we have

$$\phi(n) = n \prod_{i=1}^{r} \left(1 - \frac{1}{p_i}\right).$$

10.7 Problems

(1) Compute the following:

(a) $135^{41} \pmod{41}$
(b) $62^{108} \pmod{53}$

(2) Compute the following:

(a) $7^{26} \pmod{72}$
(b) $3^{48} \pmod{112}$

(3) There is some integer n for which

$$133^5 + 110^5 + 84^5 + 27^5 = n^5.$$

Without using a calculator or doing too much computation, determine n. (AIME 1989)

(4) (a) Without doing too much computation, show that $2^{340} \equiv 1 \pmod{341}$. (This would follow from Fermat's Little Theorem if 341 were prime, but $341 = 11 \times 31$.)
(b) Show that for any integer a with $\gcd(a, 561) = 1$, $a^{560} \equiv 1 \pmod{561}$. (Numbers with this property are called *Carmichael numbers*, and 561 is the smallest Carmichael number.)

(5) Find primitive roots modulo 17, 23, and 29.

(6) Let p be a prime, and let $a \in \mathbb{F}_p^\times$. Show that a is a primitive root if and only if $p - 1$ divides x whenever $a^x \equiv 1 \pmod{p}$.

(7) Show that, for every positive integer n,

$$\prod_{i=0}^{n-1} (2^{2^i} + 1) + 2 = 2^{2^n} + 1.$$

Use this to give a new proof that there are infinitely many primes.

(8) Compute the binary representation of 500. Compute the base-7 representation of the number whose base-5 representation is 3414.

(9) For how many integers $n \leq 2017$ does the binary representation of n contain more 1's than 0's? (Adapted from AIME 2003.)

(10) Prove the divisibility rule for 11: A number $n = abcdef\cdots$ (where $abcdef\cdots$ is its decimal representation) is divisible by 11 if and only if $a - b + c - d + \cdots \equiv 0 \pmod{11}$. Can you find a divisibility rule for 7? (Possible hint: $7 \times 143 = 1001$.)

(11) We mentioned in the chapter that the divisor function d is multiplicative, i.e., that if $\gcd(a, b) = 1$, then $d(ab) = d(a)d(b)$. Show that this relation does *not* necessarily hold if a and b are not relatively prime.

(12) Suppose f is a multiplicative function, and let

$$g(n) = \sum_{d|n} f(d).$$

(The notation $d \mid n$ means "d divides n.") Show that g is a multiplicative function.

(13) The Möbius function μ is an important function in number theory, defined as follows: if

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

is the prime factorization of n, where each p_i is a *different* prime and each $e_1 \geq 1$, we set

$$\mu(n) = \begin{cases} 0 & \text{if some } e_i \geq 2, \\ (-1)^r & \text{otherwise.} \end{cases}$$

That is, $\mu(n) = 0$ if n is divisible by the square of any prime, and otherwise $\mu(n) = 1$ if n has an even number of prime factors and -1 if n has an odd number of prime factors. Show that μ is a multiplicative function.

(14) Let $f : \mathbb{N} \to \mathbb{R}$ be any function, and define

$$g(n) = \sum_{d|n} f(d).$$

Show that

$$f(n) = \sum_{d|n} \mu(d) g(n/d).$$

This is known as the *Möbius inversion formula.*

(15) Let $\zeta(s) = \sum_{n=1}^{\infty} \frac{1}{n^s}$. Show that

$$\frac{1}{\zeta(s)} = \sum_{n=1}^{\infty} \frac{\mu(n)}{n^s}.$$

Chapter 11
The Diffie–Hellman Key Exchange and the Discrete Logarithm Problem

11.1 The Diffie–Hellman Key Exchange

The goal of the Diffie–Hellman key exchange, described in [DH76], is to create a key for Alice and Bob by sending shared information back and forth a few times. The key is created from several pieces, some of which are public information, and some of which only Alice knows or only Bob knows. The point is that knowing all the public pieces and at least one of the private pieces makes it easy to construct the key, but without knowing any of the private pieces, working out the key is extremely difficult (but possible in some sense). Here is how it works:

First, Alice and Bob between them pick some prime number p. Now, as we mentioned a bit earlier, $\mathbb{F}_p^\times$ has a primitive root g. (There are a lot of them, but they just agree to choose one of them.) The two numbers p and g are public information.

Next, Alice chooses some number a, and Bob chooses some number b. They keep these numbers secret. Alice computes $g^a \pmod{p}$ and sends it to Bob (and the rest of the world). Similarly, Bob computes $g^b \pmod{p}$ and sends it to Alice (and the rest of the world). Now, Alice is in possession of the number $g^b \pmod{p}$, so she can compute $(g^b)^a = g^{ab} \pmod{p}$. Similarly, Bob is in possession of the number $g^a \pmod{p}$, so he can compute $(g^a)^b = g^{ab} \pmod{p}$. Hence, they can each work out the number $g^{ab} \pmod{p}$.

The number $g^{ab} \pmod{p}$ is the key. Alice and Bob can then use it to encrypt messages.

Let's see an example of the key exchange in action.

Example Let us take $p = 47$. The smallest primitive root modulo 47 is 5, so let's take $g = 5$. Let us suppose that Alice chooses $a = 19$, and I won't tell you what Bob chooses b to be. Alice computes $5^{19} \pmod{47}$, which is 10. Bob computes $5^b \pmod{47}$ and gets 21. Now, Alice computes $5^{ab} \pmod{47}$, which is $21^{19} \equiv 28 \pmod{47}$. So, 28 is the key.

Why does this work from a cryptographic point of view? That is, why can't Eve compute the key from the public messages that Alice and Bob send? The point is that

S. Rubinstein-Salzedo, *Cryptography,* Springer Undergraduate Mathematics Series, https://doi.org/10.1007/978-3-319-94818-8_11

it is not easy to see how to compute $g^{ab} \pmod{p}$ from knowing $g^a \pmod{p}$ and $g^b \pmod{p}$; we have to know what either a or b is in order to do that. But it isn't easy to reconstruct a or b from knowing $g^a \pmod{p}$ and $g^b \pmod{p}$. Of course, Eve knows that $g^{p-1} \equiv 1 \pmod{p}$, so she can assume that $0 \le a, b \le p-2$. But that still means she has $p-1$ numbers to check, and it's not clear if she can do substantially better than just computing $g^i \pmod{p}$ for every $0 \le i \le p-2$ and checking if it's either g^a or g^b. With $p = 47$, it won't take her too long to find a or b. But what if p is a much larger prime, say a prime with 30 digits? Now, it will take her a very long time to reconstruct a or b by trying all the possibilities!

In fact, we will soon see some ways that Eve can attack Diffie–Hellman more efficiently than just checking all the possibilities. But it still isn't fast enough.

11.2 The Discrete Logarithm Problem

Let's step out of the modulo p world for a moment and return to the more familiar world of (ordinary) integers. Suppose we have integers a and n, and we know a and a^n. Then it is easy to figure out what n is. We have a function that computes it for us: $n = \log_a(a^n)$, the base-a logarithm of a^n. But even if we didn't know how to use this function, it still wouldn't take too long to work out what n is. For example, we could start by making a guess m for what we think n should be. We then compute a^m. If it's smaller than the known value of a^n, we try a bigger value of m. If it's larger than a^n, we try a smaller value of m. In this way we can narrow down the possibilities for n until we get it right, and we'll figure it out pretty fast.

But when we work in $\mathbb{F}_p$, the situation is totally different. Since we no longer have a notion of how large elements of $\mathbb{F}_p$ are (or at least no useful notion), we can't use the size of candidate answers as a clue to whether we're too high or too low.

By analogy with the integer (or real number) case, if g and g^n are elements of $\mathbb{F}_p^\times$, we call n the *discrete logarithm* to base g of n.

More generally, if we have any cyclic group G with generator g, we call n the discrete logarithm of g^n.

It is widely believed, although not proven, that there is no polynomial time algorithm for computing discrete logarithms in general. This means that, if the size of the group (or the prime p in the case of $\mathbb{F}_p^\times$) is very large, then it will take a very, very long time to compute discrete logarithms, barring a very unlikely miracle. (Why the possibility of a miracle? Well, one possible attempt to solve the discrete logarithm problem would just be to guess values of n randomly; it could happen that the first or second random guess just happens to be right!)

Problem 11.1 (*Discrete Logarithm Problem (DLP)*) Given a cyclic group G, a generator g for G, and some element x of G, find a number n so that $x = g^n$.

Whether the discrete logarithm problem for a cyclic group G is hard depends tremendously on the form of the group. We mentioned earlier that $\mathbb{F}_p^\times$ is equivalent to $\mathbb{Z}/(p-1)\mathbb{Z}$ under addition. Yet, solving the discrete logarithm problem in $\mathbb{Z}/(p-1)\mathbb{Z}$ is very easy: if we know g and ng, then $n = (ng)g^{-1}$, where g^{-1} is the inverse of g in the *multiplicative* group $(\mathbb{Z}/(p-1)\mathbb{Z})^{\times}$. We have already seen how to compute these inverses easily using the Euclidean algorithm.

Of course, smart people have been working on the discrete logarithm problem for quite a long time, and they have come up with algorithms that are faster than just checking all the possibilities, even quite a lot faster. One such algorithm is, rather memorably, called the baby-step giant-step algorithm, and we'll look at it soon. But all known algorithms are still impractical for large p.

Furthermore, there are some groups in which the discrete logarithm problem is even harder to solve than it is in $\mathbb{F}_p^{\times}$, as there are some tricks such as the *index calculus* that work to attack the discrete logarithm problem in $\mathbb{F}_p^{\times}$ but not in a general cyclic group. Later on, we will look at elliptic curves, which allow for another version of the Diffie–Hellman key exchange (and many other things), but the discrete logarithm problem is even harder to crack for elliptic curves than it is for $\mathbb{F}_p^{\times}$.

The problem that arises in the Diffie–Hellman key exchange isn't quite the same as the DLP, but it is similar.

Problem 11.2 (*Diffie–Hellman Problem (DHP)*) Given a cyclic group G, a generator g for G, and two elements g^a and g^b of G, compute g^{ab}.

If we can solve DLP, then it is easy to solve DHP. It might be that it is *easier* to solve DHP than DLP, but we do not know. So far, no one has proposed an algorithm for solving DHP that does not rely on first solving DLP, and it is widely believed that none exists.

11.3 ElGamal Encryption

The Diffie–Hellman key exchange is not, strictly speaking, a cryptosystem, as it doesn't provide any mechanism for encrypting messages. Rather, it just provides a mechanism for establishing one particular piece of shared secret information. There are various ways of constructing actual cryptosystems based on the Diffie–Hellman key exchange, perhaps the most famous being ElGamal encryption, as introduced in [Elg85]. Here is how it works. In this setup, it will be Bob sending a message to Alice.

First, Alice generates a key. To do this, she picks a prime p and a generator g of $\mathbb{F}_p^{\times}$. Then she chooses a random a between 0 and $p-2$. Alice then computes $h = g^a \pmod{p}$. Alice publishes h, g, and p. This is her public key. She keeps a secret.

Now, it's Bob's turn to do something. He chooses a random b between 0 and $p-2$, and he computes $s = h^b \pmod{p}$. Now, Bob has a plaintext message m he wishes to send. We will assume that m is an element of $\mathbb{F}_p^{\times}$; if it is not a number but a sequence of words, then he must first convert it to an element of $\mathbb{F}_p^{\times}$. Now, Bob calculates the ciphertext, which consists of both $g^b \pmod{p}$ and $c = ms \pmod{p}$.

He then sends both $g^b \pmod{p}$ and c to Alice. Note that the ciphertext consists of *two* numbers: $g^b \pmod{p}$ and c.

So, that's the encryption part of ElGamal. Now, when Alice receives g^b and c, she needs to be able to recover the plaintext. In order to do so, she first calculates s, which she can do easily since it is $(g^b)^a \pmod{p}$, and she knows both g^b and a. Now that she knows s, she can use the Euclidean algorithm to compute s^{-1} quickly, and then she can compute $m = s^{-1}c \pmod{p}$ and recover the plaintext message.

Of course, just as in the Diffie–Hellman case, Eve has a lot of trouble computing s just from knowing $g^a \pmod{p}$ and $g^b \pmod{p}$.

Let us go through an example of an encryption using ElGamal.

Example Let us this time take $p = 83$ and $g = 2$, which is a primitive root. Alice chooses a to be 51 and computes $h = g^a \pmod{p}$, which is 55. So, she sends p, g, and h to Bob and the rest of the world.

Now, Bob has to choose some number b, and let's say he chooses 38. So, he has to compute g^b and h^b. He finds that $g^b \equiv 31 \pmod{83}$ and $s = h^b \equiv 27 \pmod{83}$. Suppose that the message he wants to encrypt is $m = 44$. Then he computes the ciphertext to be $c = sm = 26$. So, Bob sends $c = 26$ and $g^b = 31$ back to Alice and the rest of the world.

Now, Alice has to compute m from c. To do this, she first computes s, which is $(g^b)^a = 31^{51} \equiv 27 \pmod{83}$. Now, she has to find the inverse of 27 modulo 83. Just because this is such an important step, we'll recall how to do it. We need to find numbers k and n so that $27k + 83n = 1$. (Actually we only need to find k, but let's do both anyway.) We do this by going through the Euclidean algorithm. The first step of the Euclidean algorithm will replace 83 with $83 \pmod{27}$, which is 2. We have

$$2 = 83 - 3 \times 27.$$

The next step replaces 27 with $27 \pmod{2}$, which is 1. We have

$$1 = 27 - 13 \times 2.$$

But we know how to write 2 in terms of 83 and 27, so we have

$$\begin{aligned} 1 &= 27 - 13 \times 2 \\ &= 27 - 13(83 - 3 \times 27) \\ &= -13 \times 83 + 40 \times 27. \end{aligned}$$

Hence $k = 40$.

Thus, the original plaintext message is $m = s^{-1}c = kc = 40 \times 26 \equiv 44 \pmod{83}$. And indeed, this was the original plaintext message that Bob wanted to send.

So, how hard is it to crack ElGamal? The answer is that it's as hard as cracking Diffie–Hellman. That is, if we can crack ElGamal, then we can also crack the DHP, and vice versa. The following theorem is very typical of the sort of results people work on in cryptography. It involves assuming that Eve has access to an *oracle* that can, through some magical process, tell her answers to questions of certain very specific types, very quickly. In this case, we assume that she has access to an oracle that can provide her with quick decryptions of ElGamal ciphers, but that cannot do anything else.

Theorem 11.3 *Fix p and g. Suppose that Eve has an oracle that decrypts ElGamal with arbitrary ciphertexts and public keys. Then she can use the oracle to solve DHP.*

Proof Given $A = g^a \pmod{p}$ and $B = g^b \pmod{p}$, she needs to compute $g^{ab} \pmod{p}$. She hands the oracle B for g^b and 1 for c, and the oracle tells her that the plaintext message is $(B^a)^{-1} = g^{-ab} \pmod{p}$. She can then take the inverse of this number to obtain $g^{ab} \pmod{p}$. ■

11.4 Symmetric Versus Asymmetric Key Cryptography

There is something fundamentally different about the sort of ciphers like the Vigenère we saw earlier and the modern cryptosystems like ElGamal and the others we'll be seeing soon, in the way that they are encrypted and decrypted. When we use the Vigenère cipher or one of the other ciphers, the same key is used to encrypt and decrypt messages. With ElGamal encryption, it takes a different key to encrypt and decrypt messages.

Here is one amusing consequence of this fact: suppose Alice sends Bob a message using a Vigenère cipher, but then she forgets what the message is and wants to reconstruct it. As long as she still has the key, she can decrypt it.

However, if Bob sends Alice a message using ElGamal and forgets the message, he is out of luck. Alice and Bob didn't really share a key in ElGamal, or not exactly. If Bob remembers what b is, he can reconstruct the message easily, but otherwise he can't, since he has no more information than Eve does.

Or, suppose that Bob does remember what b is and wants to reconstruct the message. His computation to reconstruct the message is *different* from what Alice must do to read the message: Alice computes m as $g^{ab}m/(g^b)^a$, whereas Bob computes it as $g^{ab}m/(g^a)^b$. This is the mark of an asymmetric key cryptosystem: Alice and Bob have to do different things in order to read the message: they have different keys to decrypt it for them!

11.5 Shanks's Baby-Step Giant-Step Algorithm

The reason that we believe that Diffie–Hellman and ElGamal are hard for Eve to break is that it should take a long time to solve the discrete logarithm problem. But how long? In order to answer that, we need to find an algorithm that solves it and analyze how long it takes, in terms of big O notation. One obvious thing that Eve can try is to make guesses for a until she gets it right. There are $p-1$ possible values of a, so on average she has to guess about half of all the possibilities before she gets it right. (She might get lucky and guess it the first time, or she might get unlucky and have to go through all the possibilities before getting it, but $\frac{p-1}{2}$ is the average.)

But mathematicians are always hard at work coming up with more devious algorithms that solve the discrete logarithm problem and other similar problems more quickly. It may be surprising that it is possible to find substantial improvements, but this does turn out to be the case. One of the best-known general algorithms is called the baby-step giant-step algorithm [Sha71]. Here is the setup:

Theorem 11.4 *Let G be a cyclic group of size n with generator g. Given $h \in G$, the following algorithm computes an m so that $h = g^m$ in $O(n^{1/2+\varepsilon})$ time:*

- *Compute and store the values $1, g, g^2, \ldots, g^{\lfloor\sqrt{n}\rfloor-1}$. (These are the baby steps.)*
- *Compute $hg^{-i\lfloor\sqrt{n}\rfloor}$ for $i = 1, 2, \ldots$, and check, for each i, whether the result is on the baby step list. (These are the giant steps.) If a match is found, stop; otherwise continue to the next value of i.*

Once you have found a match, you have found i and j so that $hg^{-i\lfloor\sqrt{n}\rfloor} = g^j$, so $h = g^{i\lfloor\sqrt{n}\rfloor+j}$.

The idea is to show that there must be some match with $0 \le i \le \lfloor\sqrt{n}\rfloor + 1$. To see this, let us suppose that $h = g^m$. Write $m = i\lfloor\sqrt{n}\rfloor + j$, with $0 \le j \le \lfloor\sqrt{n}\rfloor - 1$. Then

$$i = \frac{m-j}{\lfloor\sqrt{n}\rfloor} \le \frac{m}{\lfloor\sqrt{n}\rfloor} \le \frac{n-1}{\lfloor\sqrt{n}\rfloor} \le \frac{n-1}{\sqrt{n}-1} = \sqrt{n}+1,$$

so $i \le \sqrt{n}+1$, or, since i is an integer, $i \le \lfloor\sqrt{n}\rfloor + 1$.

One might question the running time because one has to compare every giant step to every baby step, so there should be around $\sqrt{n} \times \sqrt{n} = n$ comparisons. But actually we can do the lookup faster. One way of improving the lookup is to sort the numbers in the baby step list. Once we have done this, it is very quick to check if a number is an element of a *sorted* list. (This can be done more quickly using a *hash table*, but this does not substantially improve the overall efficiency of the algorithm in this case, since the other steps take a comparable amount of time.)

Let us see how long this takes in terms of big O notation. In order to compute the baby steps, we have to perform $O(\sqrt{n})$ multiplications, each one of which takes $O(n^\varepsilon)$ steps. (It's some power of $\log(n)$.) Thus, computing the baby steps takes $O(n^{1/2+\varepsilon})$ steps.

Next, we have to compute the giant steps. Each one of them requires only one multiplication from the previous one, since $hg^{-(i+1)\lfloor\sqrt{n}\rfloor} = hg^{-i\lfloor\sqrt{n}\rfloor} \times g^{-\lfloor\sqrt{n}\rfloor}$. Thus, computing the giant steps takes another $O(n^{1/2+\varepsilon})$ steps.

Next, we have to sort the baby steps, which takes

$$O(n^{1/2}\log(n^{1/2})) = O(n^{1/2+\varepsilon})$$

steps. Finally, for each giant step, we have to check if it is in the baby step list. Each search from a sorted list takes $O(\log(\sqrt{n}))$ steps, and there are $O(\sqrt{n})$ steps. So again, $O(n^{1/2+\varepsilon})$ steps.

This is an improvement on the naïve algorithm of just checking all the possibilities, which takes $O(n^{1+\varepsilon})$ time.

Also, note that although this takes $O(n^{1/2+\varepsilon})$ steps, which looks like a polynomial time algorithm, it actually isn't, because we're measuring in terms of the *length* of the input. Thus, it's still an exponential time algorithm, just a better one than the naïve one.

Let us run through an example of the baby-step giant-step algorithm.

Example Let's take $G = \mathbb{F}_p^\times$, with $p = 317$. Let's use $g = 41$, which is a primitive root modulo p. Let's take $h = 93$. Now, we need to compute $\lfloor\sqrt{317}\rfloor$, which is 17.

We must compute the baby steps: $1, g, g^2, \ldots, g^{16}$. When we do this, we obtain the following numbers:

$$1, 41, 96, 132, 23, 309, 306, 183, 212,$$
$$133, 64, 88, 121, 206, 204, 122, 247.$$

So, those are the baby steps. Now, let's work out the giant steps. These are of the form hg^{-17i}. With $i = 0$, we have $h = 93$, which is not one of the baby steps. With $i = 1$, we have $hg^{-17} = 181$, which is also not one of the baby steps. With $i = 2$, we have $hg^{-34} = 8$, which is also not a baby step. We continue on for a while, until we get to $i = 11$, when we obtain $hg^{-11\times 17} = 64$, which *is* one of the baby steps.

Now, where is it in the list of baby steps? It's g^{10}. Hence, we have found a collision:

$$hg^{-11\times 17} = g^{10},$$

or $h = g^{11\times 17+10} = g^{197}$. Indeed, $g^{197} = h$.

11.6 Pohlig–Hellman

Sometimes, solving the discrete logarithm problem is actually quite easy, as explained in [PH78]. This happens for $\mathbb{F}_p^\times$ when $p - 1$ has only small prime factors. Suppose, for example, we take $p = 271$ and $g = 6$ (which is a generator), and we wish to find an m so that

$$g^m \equiv 145 \pmod{p}.$$

We can do this as follows:

Observe that $p - 1$ factors as $270 = 2 \times 3^3 \times 5$. We will try to determine $m_2 = m \pmod 2$ and $m_3 = m \pmod{3^3}$ and $m_5 = m \pmod 5$.

Once we find m_2, m_3, m_5, we can use the Chinese Remainder Theorem to reconstruct m.

Here is how we compute m_2: note that, for any $r \in \mathbb{F}_p^\times$, either $r^{\frac{p-1}{2}} \equiv 1 \pmod{p}$, or $r^{\frac{p-1}{2}} \equiv -1 \pmod{p}$. Those values of r for which $r^{\frac{p-1}{2}} \equiv 1 \pmod{p}$ are the even powers of g, and those for which it's -1 are the odd powers. So, we raise 145 to the power of $\frac{p-1}{2}$ and get $145^{\frac{p-1}{2}} \equiv -1 \pmod{p}$. That means that

$$145^{\frac{p-1}{2}} \equiv (g^m)^{\frac{p-1}{2}} \equiv g^{m \cdot \frac{p-1}{2}} \pmod{p},$$

so $m \times \frac{p-1}{2}$ is odd and hence m is odd. Thus $m_2 = 1$.

Now, we do something similar for m_3, but it's a little harder because we have to work modulo 27 rather than just 3. To deal with this, we write m_3 as $m_3 = c_0 + 3c_1 + 9c_2$, where c_0, c_1, c_2 are 0, 1, or 2. To compute the c_i's, we'll need to know that $g^{\frac{p-1}{3}} \equiv 242 \pmod{p}$, and $g^{2 \cdot \frac{p-1}{3}} \equiv 28 \pmod{p}$. To find c_0, we look at $145^{(p-1)/3} \pmod{p}$, and we get 28. This means that

$$145^{\frac{p-1}{3}} \equiv g^{m \cdot \frac{p-1}{3}} \equiv g^{2 \cdot \frac{p-1}{3}} \pmod{p}.$$

Thus, $m \times \frac{p-1}{3} \equiv 2 \times \frac{p-1}{3} \pmod{p-1}$, so $m \equiv 2 \pmod{3}$. This means that $c_0 = 2$.

To work out c_1, we take 145 and divide by g^{c_0} to get 132. To work out c_1, we raise 132 to the power of $\frac{p-1}{9}$ to get 28. This means that $c_1 = 2$. Now, we divide 132 by g^{3c_1} to get 3. Now, we raise 3 to the power of $\frac{p-1}{27}$ to get 242, so $c_2 = 1$. Hence, $m_3 = c_0 + 3c_1 + 9c_2 = 17$, so $m \equiv 17 \pmod{27}$.

And now for m_5. It will be helpful to know that

$$\begin{aligned} g^{\frac{p-1}{5}} &\equiv 10, & g^{2 \cdot \frac{p-1}{5}} &\equiv 100, \\ g^{3 \cdot \frac{p-1}{5}} &\equiv 187, & g^{4 \cdot \frac{p-1}{5}} &\equiv 244. \end{aligned}$$

Now, just as before, we work out m_5 by raising 145 to the power of $\frac{p-1}{5}$, to get 1. Hence, $m_5 = 0$, and $m \equiv 0 \pmod{5}$.

From all this, we have learned that $m \equiv 1 \pmod{2}$, $m \equiv 17 \pmod{27}$, and $m \equiv 0 \pmod{5}$. This will allow us to reconstruct $m \pmod{270}$. We can use the Chinese Remainder Theorem to put these together, to see that $m \equiv 125 \pmod{270}$. Indeed, it is true that $6^{125} \equiv 145 \pmod{271}$.

When $p-1$ factors into small primes, Pohlig–Hellman is the way to go in attacking the discrete logarithm problem. However, Alice and Bob ought to know that as well, so if they do not want their message to be easily attackable, they ought not to choose a prime p for which $p-1$ has only small prime factors. This is relatively rare though: we roughly expect the largest prime factor of a number n to be something like $\frac{n}{\log(n)}$.

Now, if p is any prime larger than 2, then 2 will be a factor of $p-1$. However, it can happen that $\frac{p-1}{2}$ is prime, so that $p-1$ has a prime factor that's as large as possible. These primes are generally considered to be good for cryptography. These are called safe primes, or Sophie Germain primes. (More precisely, if p and $2p+1$

are both primes, then p is called a Sophie Germain prime, and $2p + 1$ is called a safe prime.) The largest known safe prime is

$$18543637900515 \times 2^{666668} - 1.$$

It is believed that there are infinitely many such primes, but this has not been proven.

11.7 The Index Calculus

As we will see later, in Chapter 14, it is possible to create analogs of the Diffie–Hellman/ElGamal protocol, in other finite abelian groups. In particular, we will see how to create an elliptic curve version of it. Of the attacks we have looked at so far against the discrete logarithm problem, the baby-step giant-step attack works just as well in any abelian group. The Pohlig–Hellman attack also works in an arbitrary abelian group *if* we know the number of elements in the group, and that number has only small prime factors. In the elliptic curve version, an attacker will likely be able to compute the number of elements in the group, but that number probably won't only have small prime factors; in fact, the number of elements might be a large prime. We see, therefore, that the baby-step giant-step attack is a general attack against a large family of cryptosystems, whereas Pohlig–Hellman is more specialized: sometimes it is effective, and sometimes it isn't.

Many cryptographic attacks are rather specialized, but they can specialize in various different directions. One way that an attack on the discrete logarithm problem can specialize is to work against the standard version of the discrete logarithm problem, but not against the elliptic curve version or other related types. One powerful attack of this form, that we will now discuss, is the *index calculus* attack. A similar idea will show up again when we discuss the quadratic sieve in Section 13.5.

The idea is to pick a *factor base*, which is typically chosen to be the first r primes (for some value of r): $\mathcal{F} = \{2, 3, 5, 7, \ldots, p_r\}$. Then, if we wish to find x such that $g^x \equiv h \pmod{p}$, we let $k = 1, 2, \ldots$ and see if we can factor $g^k \pmod{p}$ as $2^{e_2}3^{e_3}5^{e_5} \cdots p_r^{e_{p_r}}$, i.e., using only the primes in $\mathcal{F}$. For most values of k, this doesn't work, because $g^k \pmod{p}$ will have prime factors larger than p_r. But occasionally (and often enough) it does work. We make a list of k and the factorization of $g^k \pmod{p}$ on those occasions when it does work.

Once we have enough of these relations, we can solve a system of equations modulo $p - 1$ to determine, for each $f \in \mathcal{F}$, the discrete logarithm $\ell(f)$ of f, i.e., a number $\ell(f)$ so that $g^{\ell(f)} \equiv f \pmod{p}$.

Once that is done, we start again, but trying to factor $g^m h \pmod{p}$ over the factor base, for $m = 1, 2, \ldots$ If we find a factorization of such an element, then it is easy to compute the discrete logarithm of h.

Let's do an example, where we see how this is actually done in practice.

Example Let's take $p = 232487$. This isn't completely random: I carefully selected a safe prime, for two reasons. For one thing, safe primes are not susceptible to the last attack we looked at, the Pohlig–Hellman attack, so they need a new trick to break them. But more importantly, they make computations *easier* for the index calculus attack. This attack still works when p is not a safe prime, but it would mean a bit more computation for us for no extra insight. Let us choose $g = 229401$, which is a primitive root modulo p, and take $h = 171020$. Our goal is to use the index calculus to find an x so that $g^x \equiv h \pmod{p}$.

The first step is to choose a factor base, which I will take to be $\mathcal{F} = \{2, 3, 5, 7\}$. The immediate goal is to compute the discrete logarithms of all the primes in $\mathcal{F}$. We do this by looking for numbers i such that we can factor $g^i \mod p$ using only the primes in $\mathcal{F}$. There are quite a lot of these; we will return to this point in Chapter 13. Such numbers are called *smooth* numbers, and we define them in Section 13.4 and estimate how many there are up to x in problem 20 of Chapter 13. In our case, a quick Sage script finds that $g^i \mod p$ can be factored over $\mathcal{F}$ for $i = 482, 600, 876, 948$. In particular:

$$\begin{aligned} g^{482} &\equiv 2^5 \times 3^2 \times 7^3 \\ g^{600} &\equiv 2^2 \times 3^9 \\ g^{876} &\equiv 2^9 \times 3^4 \\ g^{948} &\equiv 2^6 \times 5^2, \end{aligned}$$

where all congruences are modulo p.

Let us write x_2, x_3, x_5, x_7 for the discrete logarithms of $2, 3, 5, 7$, i.e., x_2 is the number (modulo $p - 1$) such that $g^{x_2} \equiv 2 \pmod{p}$, and similarly for the others. The first equation, for g^{482}, tells us that

$$g^{482} \equiv (g^{x_2})^5 \times (g^{x_3})^2 \times (g^{x_7})^3,$$

which we can rewrite as

$$g^{482} \equiv g^{5x_2+2x_3+3x_7} \pmod{p},$$

or, since g is a primitive root modulo p,

$$482 \equiv 5x_2 + 2x_3 + 3x_7 \pmod{p-1}.$$

In the same way, we get the following system of four equations (or congruences) in four variables:

$$\begin{aligned} 5x_2 + 2x_3 + 0x_5 + 3x_7 &\equiv 482 \\ 2x_2 + 9x_3 + 0x_5 + 0x_7 &\equiv 600 \\ 9x_2 + 4x_3 + 0x_5 + 0x_7 &\equiv 876 \\ 6x_2 + 0x_3 + 2x_5 + 0x_7 &\equiv 948, \end{aligned}$$

where all congruences are taken modulo $p - 1$. This is a system of four equations in four variables, so we expect to be able to find a solution, at least most of the time. (On rare occasions, some of the equations may be redundant, in which case we have to get another one.)

There is a standard linear algebra technique to solve systems of equations like this quickly, known as *Gaussian elimination*, or *row reduction*. However, it works much better over $\mathbb{Z}/p\mathbb{Z}$ when p is prime than over $\mathbb{Z}/n\mathbb{Z}$ when n is composite. Indeed, Sage throws an error when I try to do this computation modulo $p - 1$ directly. This is where we use the fact that p is a safe prime: we solve for x_2, x_3, x_5, x_7 modulo the prime factors of $p - 1$, and then we use the Chinese Remainder Theorem to piece them back together. To solve for x_2, x_3, x_5, x_7 modulo $\frac{p-1}{2} = 116243$, I type the following Sage code:

```
A = matrix(Zmod(116243),[[5,2,0,3,482],
    [2,9,0,0,600],[9,4,0,0,876],[6,0,2,0,948]])
A.rref()
```

Sage then returns

```
[     1      0      0      0 38292]
[     0      1      0      0 30305]
[     0      0      1      0  1841]
[     0      0      0      1 71128]
```

This is to be interpreted as saying that

$$x_2 \equiv 38292, \qquad x_3 \equiv 30305, \qquad x_5 \equiv 1841, \qquad x_7 \equiv 71128,$$

where all congruences are modulo 116243.

Next, we have to do the same thing modulo 2. When we do that, then Sage returns

```
[1 0 0 0 0]
[0 1 0 0 0]
[0 0 0 1 0]
[0 0 0 0 0]
```

This tells us that $x_2, x_3, x_7 \equiv 0 \pmod 2$, but it doesn't tell us about x_5. What that means is that we cannot tell what x_5 is modulo 2, but there are only two choices, so we aren't too concerned.

Now, we piece them together with the Chinese Remainder Theorem: for instance, we have

$$x_2 \equiv 38292 \pmod{116243}, \qquad x_2 \equiv 0 \pmod 2,$$

which means that $x_2 \equiv 38292 \pmod{p-1}$. With x_5, we have two choices, so we just test both of them by raising g to those powers and checking which one is actually

5 (mod p). Once we have done this, we find that

$$\begin{aligned} x_2 &\equiv 38292 \\ x_3 &\equiv 146548 \\ x_5 &\equiv 1841 \\ x_7 &\equiv 71128. \end{aligned}$$

Note that we could do something similar modulo an arbitrary prime p, as long as we are able to factor $p-1$. If $p-1$ contains any repeated prime factors, say q^e then, we also have to work a little harder to solve the equations modulo q^e, but in principle, this is only a minor additional difficulty.

So far, none of this has anything to do with h, so now it's time to bring that into the game. What we do now is to look for some j so that $g^j h$ is factorable over $\mathcal{F}$: any such j will do. By testing all the possibilities until we find one, we observe that $j = 166$ fits the bill: we have

$$g^{166}h \equiv 2^7 \times 3^2 \times 5^2 \times 7 \pmod{p}.$$

Let us suppose now that $h \equiv g^x \pmod{p}$. Then, writing everything in terms of powers of g, we have

$$g^{166+x} \equiv g^{7x_2+2x_3+2x_5+1x_7} \pmod{p},$$

so that

$$166 + x \equiv 7x_2 + 2x_3 + 2x_5 + x_7 \pmod{p-1}.$$

It follows that

$$x \equiv 7x_2 + 2x_3 + 2x_5 + x_7 - 166 \pmod{p-1},$$

and since we have already worked out the values of x_2, x_3, x_5, x_7, we can plug them in to get $x = 170812$. And we can verify that this does indeed work: $g^{170812} \equiv h \pmod{p}$, as desired!

The index calculus attack needs a factor base in order to get going, and this in turn relies on our ability to find a bunch of elements in the group that can serve as primes. In other groups, such as those coming from elliptic curves, there is no obvious way of doing this, and so the index calculus attack fails.

The running time of the index calculus attack, in terms of big O notation, is substantially faster than the baby-step giant-step algorithm: it is $O(e^{\log(n)^{1/2+\varepsilon}})$. If you find such an expression hard to parse, recall that the obvious "check all the possibilities" approach takes $O(n) = O(e^{\log(n)})$ steps (in fact even a little bit more than that, because each multiplication also takes some number of steps that grows with n, so let us say $O(e^{(1+\varepsilon)\log(n)})$ steps). By contrast, the baby-step giant-step algorithm takes $O(n^{1/2+\varepsilon}) = O(e^{(1/2+\varepsilon)\log(n)})$ steps. These are *much* larger than $O(e^{\log(n)^{1/2+\varepsilon}})$, so the index calculus attack is better than these other ones, in the situations where it applies. This attack is part of the motivation for using elliptic curves in cryptography: they don't seem to suffer from as many specialized attacks.

11.8 Problems

(1) Alice and Bob decide to use the Diffie–Hellman key exchange with $p = 41$ and $g = 7$. Alice chooses $a = 14$, and Bob chooses $b = 23$. What is the key?

(2) Suppose you're Eve, and you know that Alice and Bob are using a Diffie–Hellman key exchange with $p = 19$ and $g = 10$. If $g^a = 8$ and $g^b = 7$, what is the key?

(3) Bob sends Alice another message encrypted with ElGamal encryption. They use $p = 79$ and $g = 3$. Alice chooses $a = 44$, and Bob returns the ciphertext $(g^b \pmod{p}, m \times g^{ab} \pmod{p}) = (74, 50)$. Decrypt the message.

(4) Suppose that Eve has an oracle to solve the discrete logarithm problem, i.e., if she tells it g and m, then the oracle tells her an x so that $g^x \equiv m \pmod{p}$. How can she use this oracle to break ElGamal encryption quickly?

(5) Using the baby-step giant-step algorithm, find an x so that $5^x \equiv 193 \pmod{503}$. (5 is a generator for $\mathbb{F}_{503}^\times$.) Feel free to use a calculator.

(6) Using the Pohlig–Hellman algorithm, find an x so that $17^x \equiv 2108 \pmod{8641}$. (17 is a generator for $\mathbb{F}_{8641}^\times$.) Feel free to use a calculator, but make sure you know how to do the Chinese Remainder Theorem step with the Euclidean algorithm, or some method other than educated guessing.

(7) Bob wishes to send Alice a message encrypted with ElGamal encryption. They decide to use $p = 73$ and $g = 5$. Alice picks a and computes $g^a \equiv 49 \pmod{p}$. Bob chooses $b = 33$ and wishes to send the message $m = 62$. What is the ciphertext he sends?

(8) Let $p = 1283$, $g = 264$, and $h = 910$. Use the index calculus to find an x so that $g^x \equiv h \pmod{p}$. (g is a primitive root modulo p.)

(9) Let p be a prime, and let $a \in (\mathbb{Z}/p\mathbb{Z})^\times$. If $a^x \equiv b \pmod{p}$, and x is the smallest nonnegative integer with this property, then we write $x = \operatorname{dlog}_a b$. (The dlog stands for "discrete logarithm.")

   (a) Show that if a is a primitive root modulo p, then $\operatorname{dlog}_a(bc) \equiv \operatorname{dlog}_a(b) + \operatorname{dlog}_a(c) \pmod{p-1}$.
   (b) If a is not a primitive root, give a counterexample to part (9a).
   (c) 5 is a primitive root modulo 23. Find $\operatorname{dlog}_5(13)$.

(10) Suppose that a is a primitive root modulo $p \geq 3$. Show that $x \in (\mathbb{Z}/p\mathbb{Z})^\times$ is a perfect square in $(\mathbb{Z}/p\mathbb{Z})^\times$ if and only if $\operatorname{dlog}_a(x)$ is even. (Observe that this is true for any a, and hence does not depend on the choice of primitive root.) How many elements of $(\mathbb{Z}/p\mathbb{Z})^\times$ are perfect squares?

(11) (a) Suppose that $p \equiv 1 \pmod{3}$ is prime, and let a be a primitive root modulo p. Show that x is a perfect cube if and only if $\operatorname{dlog}_a(x) \equiv 0 \pmod{3}$.

(b) What happens if $p \equiv 2 \pmod 3$?
(c) Generalize to perfect nth powers.

(12) Write down the Pohlig–Hellman algorithm formally, and give a proof showing that it actually works.

Chapter 12
The RSA Cryptosystem

12.1 RSA

So, we now understand one pretty secure modern cryptosystem, namely ElGamal. Is this the end of the story?

Actually, it's not. One disadvantage of ElGamal is that it makes the message twice as long, since Bob has to send *two* numbers when he only wants to encrypt one. So this is a bit inefficient.

As a result, it is more common to use other cryptosystems that don't have this problem.

Probably the most famous of all topics in cryptography is the RSA (Rivest, Shamir, Adleman) cryptosystem [RSA78]. We now describe how it works.

We begin with a key generation. RSA includes a private key and a public key, much like Diffie–Hellman and ElGamal. However, there is an important difference: in Diffie–Hellman, both Alice and Bob needed to create keys. Here, only one party needs a key. (In practice, both parties have RSA keys, and then RSA is used like Diffie–Hellman as a key exchange. However, this is not strictly necessary.)

To construct the key, Alice chooses two prime numbers p and q. These should be chosen to have roughly the same number of digits, in order to increase security. Alice then computes $n = pq$. Alice then computes $\phi(n)$, which is $(p-1)(q-1)$. Alice then chooses an integer e with $1 < e < \phi(n)$ and $\gcd(e, \phi(n)) = 1$. (In practice, people often choose $e = 2^{16} + 1 = 65537$.) Alice then computes $d = e^{-1} \pmod{\phi(n)}$.

The public key consists of n and e. The private key is d.

Now, using the key, Bob can send messages to Alice, and Alice can decrypt them. Suppose Bob wishes to send the plaintext message m, which is a number modulo n. We need m to be relatively prime to n, but in practice, this isn't a serious problem, as m would have to be a multiple of either p or q in order for this to fail, and both p and q are very large. (Actually, the mathematics still works when $\gcd(m, n) > 1$, although the proof is a little different. However, it is very bad cryptographically if

$\gcd(m, n) > 1$, since Eve will have a very easy way of decrypting the message in this case!) He computes the ciphertext $c = m^e \pmod{n}$ and sends it to Alice.

Now, Alice can decrypt c into m by computing $m = c^d \pmod{n}$.

Let us verify that this works. That is, if Bob encrypts m into c and Alice decrypts c into m, then she recovers the original message. The message Alice recovers is $(m^e)^d \pmod{n} = m^{ed} \pmod{n}$. Now, we recall that $m^{\phi(n)} \equiv 1 \pmod{n}$, so $m^r \pmod{n}$ depends only on $r \pmod{\phi(n)}$. Since $ed \equiv 1 \pmod{\phi(n)}$, we have $ed = 1+k\phi(n)$ for some integer k, so

$$m^{ed} \equiv m^{1+k\phi(n)} \equiv m \times (m^{\phi(n)})^k \equiv m \pmod{n}.$$

Hence, Alice can decrypt the message.

There's more to say about what Alice has to do. Recall that she has to find d, the inverse of e modulo $\phi(n)$. She can do this *because she knows* $\phi(n)$. However, no one else knows what $\phi(n)$ is. The security of RSA relies on it being hard to find $\phi(n)$ from knowing n.

If you know how to factor n, then it is easy for you to find $\phi(n)$, since if $n = pq$, $\phi(n) = (p-1)(q-1)$, so if we know how to factor numbers, then we can break RSA. However, n is a special number: it has exactly two prime factors, and not all numbers are like that. So, might it be easier to compute $\phi(n)$ for such n?

We don't think so, but no one knows.

Let us do an example of RSA.

Example Let $p = 71$ and $q = 89$. Then, $n = 71 \times 89 = 6319$. Thus, $\phi(n) = 70 \times 88$. Alice chooses e to be relatively prime to $\phi(n)$; let us say that she chooses $e = 53$. She then computes $d = e^{-1} \pmod{\phi(n)}$. Using the Euclidean algorithm, she finds that $d = 2557$. (Actually, I did it using Sage, but one *could* use the Euclidean algorithm.) Alice sends off her public key, consisting of $n = 6319$ and $e = 53$.

Now, Bob wishes to send her a message, and let's say he wishes to send $m = 2161$. He computes $c = m^e \pmod{n}$, and he gets $c = 3248$. He sends c to Alice. Alice then computes $m = c^d \pmod{n}$, which is 2161. Thus, she has recovered the message.

12.2 Attacking Naïve RSA

RSA as we have seen it so far, commonly known as Textbook RSA, is actually pretty bad, and it can be easily attacked under certain circumstances. Suppose that Eve doesn't know what the message is, but she knows that there are only a few possibilities. For example, maybe Bob's message can only be 45 or 92. Using this information, can she work out which one it is?

Yes, very easily. Since she knows n and e, she can encrypt both 45 and 92 on her own. One of those will be the ciphertext, and one of them won't!

This is an example of a *chosen plaintext attack*: Eve can work out how to encrypt certain plaintexts, and that helps her to read certain messages. Chosen plaintext attacks are colloquially known as *cribs*.

Another problem is that this version of RSA is *malleable*; that is, if we change m in certain predictable ways, then c also changes in predictable ways. If we replace m with $2m$, then c gets replaced by $2^e c$. Let us see why this matters. The Eve we have been considering so far is *passive*, in that she can only read messages and try to decrypt them. But let's suppose she becomes more sophisticated and can somehow trick Alice into decoding some of her own ciphertexts (but not the one that Bob sends, or else this is silly). Let us suppose that she intercepts a ciphertext c from Bob. She can compute $2^e c$ and ask Alice to decrypt it for her, into a plaintext u.

What is u? It is $(2^e c)^d = 2^{ed} c^d = 2c^d$. But c^d is the plaintext for c, also known as m. So u is just $2m$. So, Eve divides by 2 and recovers the original message.

So, maybe Alice also knows not to decrypt $2^e c$. However, there is nothing special about 2 here: Eve can pick a random number r to replace 2 and ask Alice to decrypt $r^e c$. The decrypted message is rm, and again Eve can just divide by r to recover m.

This may feel like an unrealistic attack: how can Eve compel Alice to decrypt her message? Well, one other thing that the cryptosystem allows us to do is to verify identity: perhaps Alice needs to convince Bob that she is actually Alice; see also Section 12.6. For example, perhaps Alice is a bank, and she needs to convince Bob that she is actually a bank before Bob is willing to give her his financial information. One way that she can do this is by offering to decrypt a bunch of messages that he sends her. People other than Alice won't be able to do that. However, Alice might not to be able to distinguish between Bob's messages and Eve's messages, so she has to comply with any request, except that she is smart enough not to decrypt an actual ciphertext that Bob sends.

This attack is called a *chosen ciphertext attack*.

While malleability makes it easier for an attacker capable of performing a chosen ciphertext attack to break a cipher, it is sometimes also a beneficial property and underlies the important topic of *homomorphic encryption*. The setup for homomorphic encryption is that we may want to perform manipulations on encrypted data without first decrypting it. For example, if I have a lot of sensitive data, I may wish to store it on a cloud server (such as the ones available by Amazon) rather than buying a supercomputer of my own. Since it is sensitive data, I will want to store it in its encrypted state. But when I want to perform manipulations, I don't want to have to download all the data first, decrypt it, perform the manipulations on my own computer (since I can't allow the decrypted version to make its way to the cloud), encrypt it again, and then upload the changed version, since that would defeat the purpose of using the cloud cluster. Instead, I want to be able to perform the manipulations directly on the encrypted version. Malleability allows us to do so. Recently, Craig Gentry has produced a fully homomorphic encryption (FHE) scheme, in which *any* arithmetical operation can be performed on the encrypted data. See his charming paper [Gen10] for a not-too-technical overview, with a running allegory about Alice's jewelry store.

12.3 Padding

Based on the attacks we just saw, we now know that, in order for RSA to be seriously usable, it must first be improved dramatically. In order to make the encoding more secure against these two attacks and other similar ones, we first modify the message. Typically, we do this in some *non-deterministic* way, that is, in some way that involves randomness. The reason that we do that is to avoid the first attack above, in which Eve can decipher the message if she knows what all the possible messages are, and there aren't too many of them. So, if a message can get encrypted into many possible ciphertexts, that will be a lot more secure. One simple way of doing this is to require Bob to append 30 random digits to the end of his plaintext message before encrypting it. There are 10^{30} ways of appending random digits, so Eve won't be likely to guess the one sequence of 30 digits that he actually chose. The encodings for these different modifications will all look totally different. The idea of "filling in" a plaintext message is known as *padding*.

In fact, this is not the padding mechanism that is actually used in practice. One common padding method is known as Optimal Asymmetric Encryption Padding, or OAEP. It works by making the message a bit longer and messing around with some of the digits, in a random but yet recoverable way. Let's now describe how OAEP works.

Since we usually do cryptography with computers, it makes sense to consider a message to be a string of 0's and 1's. We have a plaintext, and we'll pad it into a longer message; the longer message will have n bits, and the original plaintext m has $n - k_0 - k_1$ bits, where k_0 and k_1 are (fairly small) numbers built into the implementation of the protocol. First, we append k_1 0's to the end of m, to obtain a string with $n - k_0$ digits. Then, we take a random k_0-bit string r. We then take some function G that expands r into an $(n - k_0)$-bit string. So, now we have two $(n - k_0)$-digit numbers, and we apply an operation called xor: for each position, if the two bits are the same, we put a 0, and if they're different, we put a 1. Call this number X. We then apply some function H that reduces X to a k_0-bit number. We then xor that number with r to obtain Y, a k_0-bit number. We then concatenate X with Y. We know which part is X and which part is Y.

From here, how do we recover m? We first recover r, since we know X, Y, and H, and r is the xor of Y and $H(X)$. Then, we can recover m, since m (appended with the k_1 0's) is the xor of X and $G(r)$, and we know r already.

So, it's pretty easy to get the message back. The reason we use this or some other sort of padding is that it helps to defeat chosen plaintext attacks.

12.4 Blocking

So far, all of our discussion of public-key cryptography has been based on encrypting numbers: Alice wishes to send Bob a number, and so she performs ElGamal of RSA encryption and sends him a different number. But perhaps Alice would like to send

Bob a message that's written in English. The first step is to convert the English letters into numbers.

A popular method for doing this is using ASCII, which turns all the letters as well as numbers, punctuation marks, and other symbols into numbers from 32 to 127 (or 255 if you include certain other special symbols such as accented letters). One possible approach is to convert each character of Alice's message into its corresponding ASCII symbol, encrypt each one, and then send the resulting message.

This is a terrible idea!

The problem is that then all we have accomplished is to create a glorified substitution cipher. (However, this is less serious if we include some randomized padding.) An attacker can forget about all the RSA machinery and crack it just like another other substitution cipher. So why bother going through all the work of RSA if we're just going to end up with a weak code?

But RSA and ElGamal are capable of encoding long messages in one go. As a result, instead of encoding each character separately, it is better to combine a bunch of characters (or their numerical equivalents) into one big number and then encrypt that. The message might be too long to fit in a single chunk, so break it up into several chunks that are each as large as possible.

There are many ways of setting this up, so I will suggest a somewhat crude one, but one that is effective enough. Suppose we wish to encrypt the message

```
Cryptography is fun.
```

We first convert each character to ASCII and get

```
067 114 121 112 116 111 103 114 097 112
104 121 032 105 115 032 102 117 110 046
```

It is important that we have kept leading zeros; we need them so that each ASCII character takes up exactly three digits.

Now, what we do is to delete the spaces and glue all the numbers together, to get

```
067114121112116111103114097112
104121032105115032102117110046
```

Now, if the message recipient's public key is large enough to encrypt that number in one chunk, then we do so and send it. If not, we break it up into smaller chunks. We can discuss with the recipient in advance how long the chunks will be. (That can be done publicly, as it does not help an attacker break it, at least not enough.)

The resulting cipher is still a substitution cipher, but rather than having 26 letters, it has far, far more than that. So, while it is theoretically susceptible to frequency attacks just like the standard substitution cipher, an attacker has little chance of being able to acquire enough data to perform the attack. As we saw earlier, it's hard to break a standard substitution cipher if the message is shorter than a few hundred characters; imagine how many more would be needed to break one in which the alphabet contains 10^{50} symbols!

12.5 Why Isn't Cryptography Solved?

So far, we have seen two cryptosystems (ElGamal and RSA) that are hard to break. And we'll see an even better one, elliptic curve cryptography, soon. So, why aren't we able to send messages securely and reliably?

Consider the following thought experiment. I'm a good person, and I believe that people ought to have the right to privacy and information security. So, as a service to society, I write a program for sending messages securely, using a very strong cryptosystem. Since I have the best of intentions, I make it available for download, free of charge, and people start using it. All is going well, until one day I get a knock on my door. It's a bunch of big and scary people from the FBI. They have guns and look really mean. They tell me to introduce a security loophole in my program, and that I am not allowed to tell anyone about their visit or the changes I'm making to my program. Since the people with guns scare me, I do what they say, and now the NSA can read all the messages sent by people who are using my program, and none of my users know anything about it.

As far as I know, from a mathematical standpoint, cryptography is a solved problem.[1] Perhaps the NSA has broken RSA and ElGamal and isn't telling, but I doubt it. In math we trust, right? The remaining problems in cryptography at the moment, at least interpreted to mean sending messages which are too difficult to break, are primarily political.

12.6 Digital Signatures

Another nifty thing we can do with public-key cryptography is to sign our messages digitally. The idea is that if Bob receives a message from Alice, he wants to make sure that it's actually from Alice, and not from Eve. If Eve can impersonate Alice and send messages to Bob pretending to be Alice, then that seems like a Bad Thing.

Fortunately, RSA admits a simple digital signature scheme. It isn't necessarily the best approach to digital signatures, but it will do for now as a sort of proof of concept; at the end of the section, we'll talk about its weaknesses. In order to use RSA digital signatures, both Alice and Bob must have RSA keys. We need to define encryption and decryption functions for both of them. Suppose that Alice's encryption function is e_A, and her decryption function is d_A. (That is, to encrypt a plaintext message m to send to Alice, Bob computes $c = e_A(m)$. To decrypt c, Alice computes $d_A(c)$, so

[1] At least, for now! There are algorithms for breaking RSA and ElGamal on a quantum computer [Sho99], so if those ever get built, we will be in trouble again; see Section 18.4 for a look at how these work. Some current research on cryptography involves coming up with quantum-resistant cryptosystems. So far, the lattice-based cryptosystems starting with [Ajt96] do not have any known quantum attacks, but our state of knowledge about what is and is not possible using a quantum computer is still in its infancy.

that $d_A(c) = d_A(e_A(m)) = m$.) Similarly, let Bob's encryption function be e_B, and let his decryption function be d_B.

Now, let us suppose that Alice wishes to send a message m to Bob, and she wishes to sign it so that Bob can not only decrypt it but also verify that it was Alice, and not Eve, who sent it. Alice first computes $c = e_B(m)$, the ciphertext. She then computes $s = d_A(c)$, the *signed* ciphertext, and sends s to Bob. This is known as the *encrypt-then-sign* protocol.

To decrypt it, Bob first computes $c = e_A(s)$ and then computes $m = d_B(c)$. After he does these two decryptions, he ends up with

$$d_B(e_A(d_A(e_B(m)))) = d_B(e_B(m)) = m,$$

the original message. Furthermore, he then knows that the message is from Alice, because Alice is the only one who knows d_A: that's her private key! If Eve tries to forge a message from Alice, she won't be able to use d_A; instead, she will have to try her own version, say d'_A. But this will fail: instead of getting the original message back, Bob will get garbled nonsense.

Remark 12.1 Note that Alice and Bob could agree to a different digital signature protocol instead, namely the *sign-then-encrypt* protocol: Alice could first compute $t = d_A(m)$ and then compute $u = e_B(t)$, and then Bob could recover m by computing $t = d_B(u)$ and then $m = e_A(t)$. As far as recovering the message goes, this is fine. However, it is worse from a cryptographic perspective. See problem 8 for a drawback of sign-then-encrypt.

Remark 12.2 There is a small subtlety here. The sort of messages that Alice can send to Bob are numbers modulo n_B, where n_B is the modulus part of Bob's public key. Hence, m and c should be numbers modulo n_B. On the other hand, in order to apply d_A to c, c should be a number modulo n_A, the modulus part of Alice's public key. In order to get around this problem, we let Alice's message consist of a sequence $m_1, \ldots, m_k$ of numbers modulo n_B, and we consider them, taken together, to form a k-digit number in base n_B. Similarly, the ciphertext is a sequence $c_1, \ldots, c_k$ of numbers modulo n_B, which we consider as a k-digit number in base n_B. In order to apply d_A to the ciphertext, Alice must first convert this number to base n_A before applying d_A. Note that this may change the number of digits/strings in the message she sends, but this causes no problems.

Abstracting a bit, we find that there is nothing special about the RSA public and private keys here. If we have *any* two pairs of encryption and decryption functions e and d with $e \circ d$ and $d \circ e$ both equal to the identity, one for Alice and one for Bob, then the same protocol works.

Is the RSA digital signature scheme good? Let's analyze what we want out of a signature scheme. First, if Bob receives a message with Alice's digital signature, then he needs to be able to trust that Alice actually sent the message, rather than Eve. That is, we don't want Eve to be able to forge Alice's digital signature. There are several levels of things we could mean by that:

Completely forgable: Eve can forge Alice's signature on any message.

Selectively forgable: Eve can forge Alice's signature on a reasonable proportion of messages that she might be interested in signing.

Existentially forgable: There is some message on which Eve can create a valid version of Alice's signature.

It is very easy for Eve to create an existential forgery using this scheme. Let us write m for a message that Eve wants to sign, so that the signed version is $d_A(m)$. For existential forgery, Eve only needs to be able to construct some pair $(m, d_A(m))$. Note that she doesn't know the function d_A, because that involves Alice's private key. On the other hand, Eve can reverse engineer this process: start with some *already signed* message m' and *encrypt* it with Alice's *public* key to get $e_A(m')$. Since $d_A(e_A(m')) = m'$, Eve is able to construct the pair $(e_A(m'), m')$, which is a valid signature. In other words, Eve can construct some message on which she can forge Alice's signature.

Is this a problem? That depends on context. Most likely, the message that Eve is able to sign is gibberish, so it won't be very valuable for her to sign this sort of message: even if Bob suspected that the message came from Alice, there wouldn't be any way for him to act on it. But what if the message isn't in English, and more generally isn't intended to be human-readable? What if the message is just randomly constructed binary strings, to be used in some other context? Or randomly generated encryption keys? Then the fact that Eve is able to sign certain messages, even nonsense ones, may become a serious issue.

Another attack on RSA digital signatures comes from malleability. If Eve finds two signed message pairs $(m_1, d_A(m_1))$ and $(m_2, d_A(m_2))$, then she can construct a new one by multiplying them: $(m_1m_2, d_A(m_1m_2))$. Is this a problem? Again, only if the product of two meaningful messages is again meaningful, and that depends on the context. Most likely, the product of two meaningful messages is gibberish: suppose the messages are phrases in English, converted to numbers. When you multiply these two numbers, the resulting number is not a numerical form of a meaningful English phrase; it will just be nonsense characters. Still, the product of two meaningful messages is more likely to be a meaningful message than a random message is, so this attack is at least slightly worrying.

On the other hand, as we already saw in Section 12.2, if Eve can request that selected messages be signed, then she can produce forged signatures on meaningful messages: if she wants to sign a message m, then she can pick m_1 and m_2 with $m_1m_2 = m$ and get Alice to sign m_1 and m_2. This gives her enough information to sign m.

In order to get around some of these problems, a common solution is not to sign the message m directly, but rather a *hashed* version of m. We'll see hash functions again in Chapter 17, especially Section 17.8, but in the meantime, here's the idea of a cryptographic hash function. Given a message m, we want to produce a new string $h(m)$, with the following properties:

- It is easy to compute $h(m)$.
- Given $h(m)$, it is computationally infeasible to reconstruct m.
- More generally, given some string h, it is computationally infeasible to find *any* message m' such that $h(m') = h$, or even to determine if one exists.

- It is hard to find two messages m and m' such that $h(m) = h(m')$.

We call such a function h a *cryptographic hash function*. Almost any sufficiently complicated (but still easily computable) function taking values in a large enough set of strings is a cryptographic hash function.

Instead of signing m directly by computing $d_A(m)$, we first hash it and then sign the hashed version: $d_A(h(m))$. This solves some of the issues that we discussed above with the RSA digital signature scheme. For example, what happens if Eve tries to use Alice's public key to produce a signed message by starting with the signature and then reconstructing the message? She would need to construct a pair $(m, d_A(h(m)))$. If she starts with the signed version $m' = d_A(h(m))$, then she can apply e_A to get $e_A(m') = h(m)$, but based on the definition of a cryptographic hash function, it is difficult to reconstruct m, or indeed any m_1 with $h(m) = h(m_1)$. Thus she is stuck. The other attacks based on malleability also fail, since a cryptographic hash function is not malleable in this way.

12.7 Attacks on RSA

The most obvious way of attacking RSA is to factor the modulus n. If Eve can learn the primes p and q with $pq = n$, then she can compute the decryption exponent d. To do this, let us suppose that she knows n, p, q, and e. Recall that d is chosen such that $ed \equiv 1 \pmod{\phi(n)}$, and that $\phi(n) = (p - 1)(q - 1)$. Since she knows p and q, she can easily compute $\phi(n)$. She then uses the Euclidean algorithm to find d and a such that $ed + a\phi(n) = 1$; only d matters here, and she may disregard a.

Next chapter, we will take up the topic of factoring and the techniques from computational number theory surrounding that problem. But factoring n from scratch is the hard way of doing things, and in practice in might be much easier to break RSA, thanks to its poor use in the real world.

In their 2012 paper [LHA+12], Arjen Lenstra, et al. showed that it is possible to break RSA quite often in practice by computing lots of gcds. In order to do this, they collected 4.7 million RSA moduli and computed the gcds of each pair of moduli. Whenever the gcd of that pair was 1, they had learned nothing from that calculation. But when the gcd is greater than 1, it must be a factor of both moduli. Assuming that the moduli are not identical, the gcd must then be a prime factor of both moduli; computing the other factor is then straightforward.

Using this technique, they were able to break around 0.2% of RSA keys. In Chapter 13, we will look at the birthday problem, which will teach us that this cannot be a coincidence: that is just far too many common primes.

But if that's not happening by random chance, what is going on? In practice, RSA keys are chosen with the help of random number generators. Start with a function f that converts a natural number n into a prime $f(n)$. (For example, $f(n)$ could be the smallest prime greater than $10^{100}n$. The actual functions used are of course much less stupid, but this one will do for illustration. But it is an interesting exercise to break RSA if you know that $f(n)$ is the function just described, and $n < 10^{40}$, say.) So, the primes p and q are chosen by starting with two random numbers m and n and letting $p = f(m)$ and $q = f(n)$.

The problem is that the random number generators are simply not good enough! The possible values for n are far from appearing equally often, and if n is a common value for the random number generator, then $f(n)$ is a common prime, likely to be detected by the gcd attack. While human eyes may not be able to notice the nonuniformity of the random number generators, a large-scale attack like the one of Lenstra, et al., which the NSA is surely also doing, will certainly notice.

There are many other crafty attacks on RSA that people have come up with over the years. We refer the interested reader to [Bon99] for a survey of some of them.

12.8 Security

Now that we have looked at two of the most popular modern cryptosystems, namely ElGamal and RSA, it's time to turn to the definition of security and analyze why these cryptosystems, at least when used correctly with padding and all that, seem to be secure under this definition.

Before we state the formal definition, let's just state a couple things we expect out of a secure cryptosystem. The setup here is that we have a message m, which gets encrypted using some key (such as an RSA key, with its usual breakdown into a public key and private key) to produce a ciphertext message c. Let's say this is a message that Alice sent to Bob. An attacker Eve would most of all like to decrypt the ciphertext c into the plaintext m given only public information. But in practice, it's likely that she can get a bit of help. For example, she might be able to make a guess as to what the plaintext message m might be, perhaps based on context or some outside knowledge of Alice and Bob. Not exactly what it is (or else her work would already be done), but perhaps she can narrow it down to one of a few possibilities. We don't want her to be able to decrypt the message, even if she knows that there are only two possibilities for what the plaintext could be.

Furthermore, perhaps Eve can somehow convince someone (perhaps Bob, or some all-knowing oracle) to decrypt a few other unrelated ciphertexts for her, as we discussed in Section 12.2. If a cryptosystem is secure, then knowing a bunch of plaintext–ciphertext pairs, even specifically chosen ones, shouldn't help Eve decrypt a different message.

Let's suppose that somehow Eve has narrowed down the list of possible plaintexts m to just two, and she thinks that each one is equally likely. Let's call these two potential plaintexts m_1 and m_2. If Eve could play some trick to determine whether m is equal to m_1 or m_2, she would, of course, be completely happy. But she may also be partially happy if she can improve her odds of guessing it correctly. Perhaps, through some sequence of queries to Bob getting him to decrypt other ciphertexts, she can update her beliefs on which one m is, so that she now thinks that the plaintext is m_1 with probability 0.7 and m_2 with probability 0.3. Now, it might be the case that she has updated her beliefs, but this update has nothing to do with reality: if she guesses m_1 (and repeats the experiment many times, with many other ciphertexts to break) she still only gets it right half the time, rather than 70% as she expects. If so, that's no breach in security. But if she can actually get it right 70% of the time after doing these queries, that's a problem with the security of the cryptosystem. She shouldn't be able to do better than 50%.

Well okay, she can do *a little bit* better than 50%, in a really obvious and stupid way: she can spend a while simply trying to guess the private key, in a way that allows her to check. For example, in RSA, she can try to guess the prime factorization of the modulus. She probably won't get it right, but in the very rare cases when she does, she will certainly be able to decrypt the ciphertext correctly.

But Eve doesn't have all the time in the world to decrypt the ciphertext: she has to do so in a reasonable amount of time. For the purposes of security, and most other things in cryptography, a "reasonable amount of time" means polynomial time in the length of the key and the message. So we won't let her try to factor the RSA modulus unless she can somehow do quickly. Very rarely, she will get lucky and factor it quickly, but the rest of the time, she will make no further progress.

Okay, with those points in mind, let's state the security game. It is played by Eve and Bob. Bob has the private key and can thus decrypt messages easily. Eve wants to decrypt the message c, and let's say that she knows that the plaintext can only be one of two messages, m_1 and m_2, with each one being a priori equally likely to her. What Eve gets to do is to submit a sequence of ciphertexts to Bob, none of which is c, and he decrypts them for her. Eve is allowed to submit her ciphertexts sequentially, so she is allowed to wait until getting back the decryption of each ciphertext before submitting the next one; this allows her to adapt her strategy to the decryptions she has already received if she wants. She is allowed to play polynomially many rounds of this game, where "polynomially many" means polynomially many in the length of the key.

After polynomially many rounds of this game, and a polynomial amount of time doing computations on her own, Eve must guess whether the plaintext message m is m_1 or m_2. We say that the cryptosystem is *secure* if there is some function ε that goes to 0 as the key length goes to ∞ such that the probability that Eve guesses correctly is less than $\frac{1}{2} + \varepsilon$. Otherwise, it is *insecure*.

Are RSA and ElGamal secure? We don't know, of course. Maybe there is a polynomial time algorithm for factoring numbers or for solving the discrete logarithm problem; we don't believe this is so, but on the other hand, there are some remarkably clever approaches to both of these problems, some of which we have already seen and some of which we shall look at in Chapter 13. We don't know any way of breaking RSA other than by factoring the modulus, but once again, no one knows whether there are other methods that could break RSA that are not equivalent to factoring, and similarly for ElGamal.

There is a caveat to be made here. Recall that when using RSA, we are supposed to pad the message first, to prevent the attack based on malleability. That attack would still work if, whenever Eve presented a ciphertext to Bob, he responded with the *padded* version of the plaintext. So he must be sure never to do that: he must only report the original unpadded version.

But perhaps the real takeaway message is this: **don't make up your own cryptosystem if you are concerned about its security**. You might think you have made a secure cryptosystem, only to find that it is vulnerable to an attack like the malleability attack on unpadded RSA. How confident are you that you would have thought of that attack on your own and defended against it with a good padding mechanism? And how confident are you that you would have had the foresight to predict other sorts of clever attacks, many of which are far more subtle than that one? Always remember that there may be many very smart people who desperately want to read your encrypted messages, and they are happy to use whatever tricks they can in order to do so. While you can learn a lot by coming up with your own cryptosystems and trying to defend them against attacks, if you actually want to send messages and not have them be read, it's best to trust the professionals and use a cryptosystem that has been tried and tested. Cryptography at a practical level is not a game for amateurs.

12.9 Problems

(1) Suppose $p = 3701$, $q = 7537$, and $n = pq$. Alice chooses her encryption key to be $e = 443$.

  (a) What is her decryption key d?
  (b) Bob wishes to send the message $m = 11034007$. What is his ciphertext?
  (c) Bob sends another message, and his ciphertext is $c = 3003890$. What was his plaintext message?
  (d) Bob is bored of sending numbers that don't mean anything and decides to send a word instead. To encrypt a word, he numbers the letters from 1 (for `a`) to 26 (for `z`) and uses 0 for a space, and he converts a word of at most four letters into a number modulo n, by making the first two digits denote the first letter, the next two digits denote the second letter, and so forth (so `code` would be coded as 03150405, for example). He sends Alice the ciphertext $c = 27870438$. What was his message?

(2) Alice decides to use RSA to allow others to send her messages, and she makes her key $n = pq = 3259499$. Through a breach in security, Eve discovers that $\phi(n) = (p-1)(q-1) = 3255840$.

  (a) Determine p and q.
  (b) Suppose $e = 1101743$, and Bob sends Alice the ciphertext 2653781. What was his message? (Calculators are strongly encouraged!)

(3) Suppose Alice's public key consists of $(n_A, e_A) = (240181, 120833)$, and her decryption exponent is $d_A = 97$. Bob's public key consists of the pair $(n_B, e_B) = (289981, 11)$. Alice wishes to sign and encrypt the message `i am alice` to Bob. Explain how she does so, including the conversion of text into

numbers. (You may use any reasonable convention of your choosing, as long as it can be decrypted unambiguously.)

(4) Suppose Alice leaks her decryption key d to Eve. Instead of choosing a new n, she decides to choose a new encryption key e and a new encryption key d, but with the same value of n.

   (a) Is this safe? (That is, can Eve decrypt messages now?)
   (b) By knowing an encryption–decryption pair (e, d) for n, is it possible to factor n quickly?
   (c) Can Eve decrypt messages *without* first factoring n?

(5) Alice and Bob are such good friends that they choose to use RSA with the same n, but their encryption exponents e and f are different, and indeed, they are relatively prime. Charles wants to send the same message m to Alice and Bob. If Eve intercepts both of his messages, how can she recover the plaintext message m?

(6) Let $f(n)$ be the smallest prime greater than $10^{100}n$. Suppose we select an RSA modulus by picking two random integers m, n with $0 < m, n < 10^{40}$, and letting our primes be $p = f(m)$ and $q = f(n)$. Explain how to recover p and q from knowing $N = pq$.

(7) Suppose that several RSA users $B_1, \ldots, B_k$ all use the same very small encryption exponent e with $e \leq k$ for their RSA encryption (but with different moduli), and Alice sends them all the same message m, resulting in ciphertexts $c_1, \ldots, c_k$. If Eve intercepts all the ciphertexts, show that she can recover the original message. This highlights at least two morals: don't use small encryption exponents, and use random padding.

(8) Our digital signature was encrypt-then-sign. The reason for this is that the reverse procedure, sign-then-encrypt, admits a weakness. Suppose that Alice sends a signed message to Bob using the sign-then-encrypt procedure. Show that Bob can forward the message to a third party, Charlie, so that it appears to Charlie that Alice sent the message directly to him. Think of a scenario in which this leads to undesirable consequences.

(9) In fact, naïve encrypt-then-sign isn't very good either. Suppose Alice wishes to send a message to Bob using encrypt-then-sign, but Eve intercepts the message before Bob gets it. Show that Eve can easily replace Alice's signature with her own one, so that Bob thinks the message comes from Eve.[2]

(10) Suppose that $n = pq$ is a product of two primes. Suppose that $\gcd(a, pq) = 1$.

   (a) Show that if $x^2 \equiv a \pmod{n}$ has any solutions in $\mathbb{Z}/n\mathbb{Z}$, then it has exactly four.
   (b) If, for some $a \in \mathbb{Z}/n\mathbb{Z}$, you know all four solutions, show that you can quickly factor n.

[2]The attacks in this and the previous problem come from Don Davis's paper [Dav01].

(11) If $p \equiv 3 \pmod 4$ is prime and a is a square modulo p (i.e., there is some x so that $x^2 \equiv a \pmod p$), then how can you *quickly* find an x with $x^2 \equiv a \pmod p$?

(12) Consider the following cryptosystem, known as the Rabin cryptosystem. Pick two primes p and q and compute $n = pq$. Pick a random $b \in \{0, 1, \ldots, n - 1\}$. The public key is (n, b). To encrypt a message $m \in \mathbb{Z}/n\mathbb{Z}$, compute the ciphertext $c = m(m + b) \pmod n$. Explain how to recover m from c, given p and q. Why does it appear to be difficult for an attacker to decrypt messages without knowing p and q? (What hard problem must the attacker solve?)

(13) Can you design other secure (or secure-seeming) cryptosystems? What hard problems are they based on? Challenge your friends to find attacks.

Chapter 13
Clever Factorization Algorithms and Primality Testing

13.1 Introduction to Clever Factorization Algorithms and Primality Tests

The main theoretical way of attacking RSA, at least when it is used with best practices, is by factoring the modulus. The most obvious way of factoring a number n is to try dividing by 2, 3, 5, 7, and so on, through all the primes less than $\sqrt{n}$, until we find a factor. This seems like the only general approach to factoring numbers, and for centuries, that was what people thought. However, in the past several decades, we have come up with fiendishly clever approaches for factoring numbers, and we will study some of them.

A closely related problem is that of primality testing: given a number n, determine whether it is prime or not. At a first glance, this seems like the same problem: in order to check if n is prime, we factor it and see if there is more than one factor or not.

However, it turns out that it is easier to determine whether a number is prime or not than it is to complete the full factorization. (Of course, if n is prime, then we have already factored it, but if n is composite, we might be able to determine that it is composite without producing an actual factor.)

Primality testing is also important for cryptography, since in both RSA and Diffie–Hellman, we need primes as part of our public keys. Furthermore, everyone needs to have different primes in order for the cryptosystems to be secure, so we need a way of generating a lot of primes, and quickly. Well, there are a lot of primes out there. (The famous Prime Number Theorem tells us that the number of primes up to x is roughly $\frac{x}{\log(x)}$, so that on average we have to pick $\log(n)$ numbers around n to hit upon a prime.) But just knowing that there are lots of primes isn't enough: we have to find them. If we can quickly check whether a number is prime, then we are in business. And, surprisingly, we can!

We begin with factorization algorithms. But in order to do that, we must first think about birthdays.

13.2 The Birthday Problem

Question 13.1 *How many people must be in a room for it to be more likely than not that two people share a birthday? (Assume that all birthdays are equally likely, and ignore February 29th.)*

Answer 23.

The way to see this is to imagine people walking into the room one at a time, until two share a birthday. The room starts out being empty. Then the first person enters the room. What is the probability that no two people share a birthday?

There's only one person, so there aren't even two people who *might* share a birthday. Thus the probability at this stage is 1, or as I prefer to write, $\frac{365}{365}$.

What happens when the second person enters? There is only one day (out of 365) taken so far: the first person's birthday. So 364 of them are safe. Thus the probability that no two people share a birthday at this stage is

$$\frac{365}{365} \times \frac{364}{365}.$$

Now, what about the third person? Two birthdays are taken, so there are 363 left. Thus, the probability that no two people share a birthday among these three is

$$\frac{365}{365} \times \frac{364}{365} \times \frac{363}{365}.$$

We continue on in this way: when there are k people in the room, the probability that no two share a birthday is

$$\frac{365}{365} \times \frac{364}{365} \times \frac{363}{365} \times \cdots \times \frac{366-k}{365}.$$

The smallest value of k for which this product is less than $\frac{1}{2}$ is $k = 23$.

In general, if there were n possible birthdays, each equally likely, it would require roughly $\sqrt{2n \log(2)}$ people before we expect to get a collision. (See problem 15.) As we are civilized mathematicians, we are of course using log to denote the natural logarithm.

Now, what does this have to do with RSA or factoring? A lot, actually: it's the key idea behind a very popular factoring algorithm, known as the Pollard ρ algorithm.

13.3 Pollard ρ Algorithm

In order to crack RSA, it seems necessary to compute $\phi(n)$, which is basically the same as factoring n. So, if our goal is to crack RSA, we need to know how long it takes to factor a number.

The basic algorithm is to divide by all numbers up to $\sqrt{n}$, but this is really slow, and there are much faster and cleverer algorithms for factoring. We'll look at a few of them, the first of which is the Pollard ρ algorithm, so called because there is a picture we can draw of it, and it looks like the Greek letter ρ. It works on the same principle as does the birthday problem: if there are n possible birthdays, it would require $O(\sqrt{n})$ people before we expect two to share a birthday.

How do we use this to our advantage? The idea is that if we have some number m, it should require about $\sqrt{m}$ (in fact $\sqrt{2m\log(2)}$) random numbers before we find two that are congruent modulo m. Then, their difference is divisible by m. Hence, if m is a factor of n, we can take a bunch of random numbers and look at their differences. But we want to collect them in an organized manner, and the following algorithm helps us to do that.

The algorithm, due to Pollard [Pol75], is actually quite simple.

Theorem 13.2 *The following algorithm tends to find a factor of n in a significantly faster amount of time than the guess-and-check method:*

(1) First, choose a random polynomial function $f(x)$ that takes values in $\mathbb{Z}/n\mathbb{Z}$ and returns values in $\mathbb{Z}/n\mathbb{Z}$.
(2) Let $x = 2$ and $y = 2$.
(3) Replace x with $f(x)$ and y with $f(f(y))$.
(4) Compute $d = \gcd(|x - y|, n)$.
(5) If $d = 1$, go back to step 3. If $d = n$, then the algorithm fails, so we have to go back to step 1 and choose another random function. If $d \neq 1$ and $d \neq n$, then success! We have found a factor of n.

Remark 13.3 There is nothing special about the choice of $x = y = 2$. That's the standard convention, but anything else will work just as well.

Let's do an example:

Example Let $n = 3255097$, and let $f(x) = x^2 + 109x + 614 \pmod{n}$. (I chose f at random.) Then we compute the following:

x	y	$\gcd(\lvert x-y\rvert, n)$
836	790634	1
790634	2363712	1
1351468	1693983	1
2363712	2632573	1
1788181	1109487	1
1693983	1153950	1
1176813	539912	1
2632573	2208952	1
1059776	225484	1
1109487	760293	1
1397369	671302	1
1153950	2019444	1
1099024	1519615	1
539912	1729914	1
1525379	2587553	211

So, 211 is a factor of n. Now, we can work out $n/211$ to find another: the remaining factor is 15427. These two numbers are both prime. So, the prime factorization of n is 211×15427.

So, that went by pretty quickly! This algorithm works especially well if one of the factors p is relatively small (somewhat liberally interpreted), since we expect to find it in something like $\sqrt{p}$ steps of the algorithm. And this is exactly because of the birthday problem.

There are other algorithms which are even a bit better, and we will look at the quadratic sieve below, which for many years was the best algorithm known. (Nowadays, there are modified versions that outperform the quadratic sieve, but they are based on the same basic ideas.) However, the Pollard ρ algorithm remains a popular choice, since it is very easy to program.

13.4 Smooth Numbers

The other algorithms rely on another curious fact: there are lots of numbers all of whose prime factors are small.

Definition 13.4 We say that a number n is k-smooth if all the prime factors of n are at most k.

To illustrate this fact, I chose the number

$$n = 23948278947252$$

by typing random digits on my keyboard, and I ran a simple Sage program to check which of the numbers from n to $n + 999$ are 1000-smooth:

```
def is_smooth(n,k):
    f = factor(n)
    if f[-1][0] <= k:
        return True
    return False

for i in range(1000):
    if is_smooth(n+i,1000):
        print i
```

The program told me that $n + 173$, $n + 588$, and $n + 660$ are all 1000-smooth. For example, we have

$$n + 173 = 5^2 \times 7 \times 17 \times 19 \times 29 \times 131 \times 229 \times 487.$$

It also doesn't take very long to check if a number is k-smooth. To do so, we keep dividing by primes less than k until we either reach 1 or aren't divisible by anymore.

The shocking fact about smooth numbers is that knowing that there are many smooth numbers lying around helps us to factor numbers that are not smooth!

Also, I suppose, I ought to be honest about the `is_smooth` function I wrote above: this is a terrible way of checking if a number is smooth! (However, it was really easy to code.) This function first factors n, and then checks to see if the largest prime factor is less than k or not. This works fine for the small numbers we'll be using, but not so well for the large numbers that are used in practice. A much better way, which is just slightly more annoying to program, is to start with n, divide by 2 as many times as possible until 2 is no longer a factor, then divide by 3 as many times as possible, then divide by 5, then divide by 7, and so forth, going through all the primes less than k. (It is possible that k will be sufficiently small that we can acquire a list of primes less than k. If not, then go through all the numbers from 2 to $k - 1$, rather than just the primes.)

13.5 The Quadratic Sieve

The quadratic sieve is one of the very best algorithms for factoring, although there is a variant called the number field sieve which does a little bit better. I am particularly partial to it, since it was discovered by Carl Pomerance [Pom82], who was my postdoctoral mentor at Dartmouth in the 2012–2013 academic year; it is probably his most important theorem.

Before we give the algorithm (in fact, a somewhat less optimized but simpler version, due to Dixon [Dix81]), let us show the idea. Suppose we had to factor 899. (Carl Pomerance himself prefers to use 8051 as an example, with the same idea.) We might notice that

$$899 = 900 - 1 = 30^2 - 1^2,$$

so it is a difference of two squares. Now, we know that differences of squares factor: $x^2 - y^2 = (x - y)(x + y)$. Hence

$$899 = (30 - 1)(30 + 1) = 29 \times 31,$$

and *voilà!* we have factored 899. So, the conclusion is that if we can find two squares whose difference is n, we can factor n (unless the squares are of numbers that differ by 1).

We can be a little more sophisticated and observe that if we find two squares that differ by a *multiple* of n, we're still doing okay: if $x^2 - y^2 = kn$, then we can look at $\gcd(x - y, n)$, and this will be a nontrivial factor of n, unless we've been stupid and chosen $x - y$ to be 1, or we are unlucky and both of them happen to be multiples of n. In the latter case, just try again; this situation is sufficiently rare that we'll get lucky and find a factor with high probability.

The question of how we find two squares whose difference is divisible by n remains, and the method uses smooth numbers. What we do, first of all, is to pick a smoothness bound B, so we're interested in B-smooth numbers. Suppose that there are $\pi(B)$ prime numbers up to B. Now, we want to find many numbers a_i so that $b_i = a_i^2 \pmod{n}$ are B-smooth; in fact, we want to find $\pi(B) + 1$ of them.

What is the point of this? Well, the a_i^2's are all squares, and so if we multiply any of them together, we still get a square. If some product of b_i's is also a square, then we've just found two squares that are congruent modulo n. While in general, it is not easy to take a bunch of numbers given to us by some process and multiply some subset of them to get a square, the situation here is not so bleak. To see why, note that a positive integer z is a square if and only if every prime factor p that divides z divides it an even number of times, so that there is some e for which $p^{2e} \mid z$ but $p^{2e+1} \nmid z$. Since there are not many prime factors available among the b_i's, it should not be too difficult to multiply some of them together in order to get a square.

So, how do we find which b_i's to take? What we do is to factor each b_i (which is easy, because they're all smooth). What does the factorization of b_i look like? It looks like

$$b_i = \prod_{j=1}^{\pi(B)} p_j^{e_{ij}}.$$

Now, we put all the e_{ij}'s, modulo 2, into a matrix. By linear algebra, we can find some subset of the rows to add to get all 0's, modulo 2. Multiply those b_i's, to get a square. Then, the product of the corresponding a_i^2's is congruent modulo n, so we have found two squares that are congruent modulo n.

So, all we're left with is the linear algebra step. First, we'll present the algorithm, and then we'll discuss what it is doing, leaving a more thorough discussion for a book on linear algebra. What we do is to put the exponents e_{ij}, modulo 2, in a matrix, flip it around, and do an operation called row reduction. We can then read off the answer.

What is row reduction? The goal is to do various operations to the rows to reduce it to one of the following form (this is over $\mathbb{F}_2$, which is slightly simpler than the case for other fields, but not much):

(1) Any row that isn't all 0's comes above any row of all 0's,
(2) The first 1 in row i comes to the right of the first 1 in row $i - 1$,
(3) If the first 1 in row i is in column j, then every other entry in column j must be 0.

How do we get a matrix into that form? We're allowed to do the following two operations:

(1) Replace row i with the sum of row i and row j,
(2) Switch two rows.

By going through these steps many times, we can eventually convert the matrix to one of the above form, and we get the same row-reduced matrix regardless of which steps we choose and the order we perform them.

Now, how can we use this to figure out which b_i's multiply to a square? Look for a column that doesn't contain the first 1 in any row. If it doesn't have any 1's in it, then we're in a lot of luck: if it's in column j, then just take b_j, and it's already a square, and we're done!

If there is at least one 1 in that column, say column j, then look for all the columns that have the first 1 in one of the rows in which column j has a 1. Multiply together all those b's, together with b_j, and that's the one to take. We also multiply together the corresponding a's, and there we have two squares which are congruent modulo n.

Now, take the gcd of the difference of their square roots and n, and this will probably be a factor of n.

Let's do an example.

Example Let's factor $n = 2968327$ using the quadratic sieve. Let's pick the smoothness bound $B = 20$, so that computations won't be too bad. Let's go past $\lfloor\sqrt{n}\rfloor = 1722$, so we'll start at 1723. We look for numbers $m \geq 1723$ so that $m^2 \pmod{n}$ is a square. Since there are 8 primes up to 20, we need to find 9 numbers m such that $m^2 \pmod{n}$ is 20-smooth, and the first 9 are

$$2467, 2512, 4567, 4585, 4607, 4633, 4711, 4857, 4921.$$

These were found by running the following Sage code:

```
n = 2968327

for i in range(5000):
    if is_smooth((1723+i)^2 % n,20):
        print 1723+i
```

We have

$$2467^2 \pmod{n} = 5 \times 11^2 \times 13 \times 19.$$

Since we're only interested in the exponents modulo 2, we keep $5 \times 13 \times 19$. But actually what we want to do is to make a matrix with the exponents modulo 2, and we obtain

$$\begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \end{pmatrix},$$

with the first entry denoting the exponent of 2, the second denoting the exponent of 3, the third denoting the exponent of 5, and so forth. Continuing in this way down the list, we obtain the full matrix

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

Now, we have to find some subset of the rows that sum to 0 modulo 2. We flip to obtain

$$M^T = \begin{pmatrix} 0&1&1&1&0&0&0&0&0\\ 0&0&0&0&1&1&1&0&0\\ 1&1&0&0&1&0&0&1&1\\ 0&0&0&1&0&0&1&0&0\\ 0&0&1&0&1&1&0&0&0\\ 1&1&0&0&0&1&1&1&1\\ 0&1&0&0&0&0&0&0&0\\ 1&0&0&0&0&0&0&0&0 \end{pmatrix}.$$

Next, perform row reduction. There are many ways of doing this, but one possibility is to start by swapping the first and third rows to obtain

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

After finishing the row-reduction process, we get

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

There are various columns that do not contain the first 1 in some row. Let us take the seventh. It has 1's in positions 3, 4, and 6. Thus, we use columns 3, 4, 6, and 7. These correspond to the third, fourth, sixth, and seventh of our nine 20-smooth numbers. We multiply those together. This means that

$$(4567^2 \mod n) \times (4585^2 \mod n) \times (4633^2 \mod n) \times (4711^2 \mod n)$$

should be a square, and indeed it is. Thus, if we let

$$r = 4567 \times 4585 \times 4633 \times 4711$$

and

$$s^2 = (4567^2 \mod n) \times (4585^2 \mod n) \times (4633^2 \mod n) \times (4711^2 \mod n),$$

then $r - s$ will not be relatively prime to n. Indeed, $\gcd(r - s, n) = 1949$, so 1949 is a factor of n. The other factor is $n/1949 = 1523$. And *voilà*! We have factored n.

So, what was all that about? Originally, each row designated a prime: the top row was about the exponent of 2 in each b_i, the second row was about the exponent of 3, and so forth. We want to find a collection $c_1, \ldots, c_r$ of columns so that the sum of the c_i's has only even entries, or, equivalently, is $0 \pmod 2$. It's hard to find them at the beginning, so we perform a bunch of operations. However, these operations do not change whether the sum of a bunch of columns will be $0 \pmod 2$ or not. (Exercise: check this!) So, we do a bunch of operations that makes it easy to find the desired columns. As we do these row operations, the meanings of the rows change: they are no longer referring to individual exponents but rather sums of exponents in the prime factorization. But that doesn't matter; we didn't really care about these primes in the first place anyway. All that we wanted was to find a square, in any way possible.

13.6 The Miller–Rabin Primality Test

We now switch topics to primality testing: given a number n, determine whether it is prime. One good approach is based on Fermat's Little Theorem. Recall that Fermat's Little Theorem says that if p is prime and a is not a multiple of p, then $a^{p-1} \equiv 1 \pmod p$. A consequence of Fermat's Little Theorem is, well, the opposite of a primality test, namely, a compositeness test: if $a^{n-1} \not\equiv 1 \pmod n$, then n is composite.

However, it sometimes happens that $a^{n-1} \equiv 1 \pmod n$ even when n is composite. For example, $2^{340} \equiv 1 \pmod{341}$. Worse yet, if $\gcd(a, 561) = 1$, then $a^{560} \equiv 1 \pmod{561}$. Numbers like 561 are called Carmichael numbers. Another of Carl

Pomerance's major theorems (together with Red Alford and Andrew Granville) is that there are infinitely many Carmichael numbers [AGP94].

Still, there is a primality test lurking nearby, known as the Miller–Rabin test [Mil76, Rab80]. Suppose we pick some number a and find that $a^{n-1} \equiv 1 \pmod{n}$. That doesn't mean that n is prime, but it means we should look more closely. What is $a^{\frac{n-1}{2}}$? (Assume that n is odd of course, as it is a very easy exercise to tell whether an even number is prime.) If n is prime, then it had better be either $1 \pmod{n}$ or $-1 \pmod{n}$: the only square roots of 1 modulo a prime are ± 1. (See problem 13.) If $a^{\frac{n-1}{2}} \not\equiv \pm 1 \pmod{n}$, then n must be composite. Next, if $a^{\frac{n-1}{2}} \equiv 1 \pmod{n}$ and $n \equiv 1 \pmod{4}$, then what about $a^{\frac{n-1}{4}}$? If $a^{\frac{n-1}{4}}$ is not $\pm 1 \pmod{n}$, then we have found a number (namely $a^{\frac{n-1}{4}}$) whose square is $1 \pmod{n}$, but that is not itself $\pm 1 \pmod{n}$, implying that n is composite.

More generally, suppose that $n - 1 = 2^e m$, where m is odd. If n is prime, then there are two possibilities: either $a^m \equiv 1 \pmod{n}$ (in which case $a^{2^d m} \equiv 1 \pmod{n}$ for any d as well), or else there is some d with $0 \le d < e$ such that $a^{2^d m} \equiv -1 \pmod{n}$. (That is, we keep taking square roots of 1; either we get 1 forever, or at some point we get -1.)

However, if n is odd and composite, and again $n - 1 = 2^e m$, then there is an extra possibility: that for some d with $0 \le d < e$, we have

$$a^{2^d m} \not\equiv \pm 1 \pmod{n}, \quad \text{but} \quad a^{2^{d+1} m} \equiv 1 \pmod{n}.$$

Alternatively, we can rephrase this:

$$a^m \not\equiv 1 \pmod{n}, \text{ but } a^{2^d m} \not\equiv -1 \text{ for any } d \text{ with } 0 \le d < e.$$

That is, we look at the numbers

$$a^m \pmod{n}, a^{2m} \pmod{n}, \ldots, a^{2^d m} \pmod{n}.$$

If $a^m \equiv 1 \pmod{n}$, then we haven't proven anything. If the sequence jumps from -1 to 1, then we also haven't proven anything. (But more on that later.) But when the sequence jumps from something other than -1 to 1 in one step, then we can be *certain* than n is composite.

When the number a allows us to prove compositeness of n, either because we jump from something not congruent to ± 1 to 1, or because $a^{n-1} \not\equiv 1 \pmod{n}$, we say that a is a (Miller–Rabin) *witness* for the compositeness of n.

Example Let us show that 57 is composite.[1] With most values of a, we will find that $a^{56} \not\equiv 1 \pmod{57}$, so then we will already know that 57 is composite.

But maybe we get unlucky and choose $a = 20$; indeed, $20^{56} \equiv 1 \pmod{57}$. Now, we start dividing the exponent by 2: we find that

[1]There is a famous story about Alexander Grothendieck and the primality of 57. See [Jac04] for one telling of the story, and some speculation about it.

$$\begin{aligned}20^{28} &\equiv 1 \pmod{57},\\ 20^{14} &\equiv 1 \pmod{57},\\ 20^{7} &\equiv 20 \pmod{57}.\end{aligned}$$

And we win! 20 is a witness to the compositeness of 57.

Note that this test did not help us at all with actually factoring 57; it only told us, nonconstructively, that is *must* be composite.

Example There are also nonwitnesses. 703 is a composite number. However,

$$3^{702} \equiv 1 \pmod{703}, \qquad 3^{351} \equiv -1 \pmod{703}.$$

If a is a nonwitness for n, then we say that n is a *strong pseudoprime* for base a.

It turns out that there are many Miller–Rabin witnesses for odd composite numbers n:

Theorem 13.5 *If $n > 1$ is odd and composite, then at least 75% of the numbers from 1 to $n - 1$ are Miller–Rabin witnesses for the compositeness of n.*

This result is nontrivial. See, for instance, [Con] for a proof.

As a result, if we want to test for the primality of n, we can pick a random a with $1 \leq a \leq n - 1$ and perform the Miller–Rabin test. If a is a witness, then we know n is composite, and if a is a nonwitness, then we don't know. But if we keep picking more and more a's, and every time we get nonwitnesses, then we increase our confidence that n is actually prime. If n is actually composite, then the chance of getting a nonwitness is at most $\frac{1}{4}$, so once we have picked k numbers, the chance of getting a nonwitness is at most $\frac{1}{4^k}$, which quickly becomes very small.

However, there are occasional numbers for which all small primes are nonwitnesses. See [SW17] for the smallest examples.

While the Miller–Rabin test is probabilistic, it can be turned into a deterministic test—if we assume the Generalized Riemann Hypothesis (GRH)! Eric Bach has shown that, assuming the GRH, if n is an odd composite number, then there is a witness a such that $a \leq \lfloor 2(\log(n))^2 \rfloor$. As a result, instead of picking values of a at random, we just start from $a = 2$ and go until we either find a witness or reach $\lfloor 2(\log(n))^2 \rfloor$; if no witness is found in that range, then n is prime. The GRH is widely believed to be true.

13.7 The Agrawal–Kayal–Saxena Primality Test

The first unconditional (i.e., not relying on the GRH or any other unproven result) polynomial-time primality test was due to Agrawal, Kayal, and Saxena in 2002 [AKS04]. The key idea is the following Lemma:

Lemma 13.6 *Let a be such that* $\gcd(a, n) = 1$. *Then, n is prime if and only if* $(x + a)^n \equiv x^n + a \pmod{n}$.

Remark 13.7 In Lemma 13.6, the two sides are polynomials in x. We say that two polynomials $f(x)$ and $g(x)$ with integer coefficients are congruent modulo n if, for every nonnegative integer k, the coefficients of x^k in $f(x)$ and $g(x)$ are congruent modulo n.

The proof is left for problem 14.

Lemma 13.6 *looks* like a primality test, but it isn't a very good one: in order to check whether or not n is prime, we have to compare roughly n coefficients and see if they all match. This is way too slow!

However, we can be cleverer. We want to show that all the coefficients of $(x + a)^n - (x^n + a)$ are divisible by n. If so, this will remain true when we group together some of the terms. Let us write $f_a(x) = (x + a)^n - (x^n + a)$, and suppose we have

$$f_a(x) = \sum_{k=0}^{n-1} b_k x^k.$$

What we will do is group together every rth term, for some well-chosen r, and see if the sum of all those coefficients is divisible by n. That is, for $0 \le c < r$, we let

$$h_a(c) = \sum_{\substack{0 \le k \le n-1 \\ k \equiv c \pmod{r}}} b_k.$$

If, for some choice of c and r, $h_a(c) \not\equiv 0 \pmod{n}$, then n *must* be composite: if n is prime, then by Lemma 13.6 $h_a(c)$ is a sum of a bunch of numbers, all of which are multiples of n, so $h_a(c)$ must also be a multiple of n.

The Agrawal–Kayal–Saxena (AKS) algorithm works by making a careful choice of values for r. Here it is, in all its glory:

Theorem 13.8 (Agrawal–Kayal–Saxena) *The following algorithm checks whether or not a number n is prime in polynomial time (in the number of digits of n):*

- *Check if n is a power, i.e., whether there exist integer $a, b > 1$ such that $n = a^b$. If so, n is composite. Otherwise, continue to the next step.*
- *Find the minimal r such that the order of r modulo n is greater than* $\log(n)^2$. *(The order of r modulo n is the least positive integer k such that $r^k \equiv 1 \pmod{n}$.)*
- *For each a from 1 to r, check whether $1 < \gcd(a, n) < n$. If this ever happens, then n is composite. (We've just found a factor, after all!) Otherwise, move on the next step.*
- *If $r \ge n$, then n is prime. Otherwise, move on to the next step.*
- *For each a from 1 to $\lfloor \sqrt{\phi(r)} \log(n) \rfloor$ and each c from 0 to r, check if $h_a(c) \equiv 0 \pmod{n}$. If this is ever false, then n is composite. Otherwise, go on to the next step.*

- *If we have made it here, then n is prime.*

  See [AKS04] for a proof.

13.8 Problems

(1) Which of the following numbers are 20-smooth?

(a) 105	(c) 309	(e) 324	(g) 529
(b) 224	(d) 323	(f) 401	

(2) Find a nonempty subset S of $\{70, 96, 120, 135, 189\}$ such that the product of the elements of S is a perfect square.

(3) (a) Suppose that a composite number n satisfies the Fermat test to base a, i.e., $a^{n-1} \equiv 1 \pmod{n}$. Show that n satisfies the Fermat test to base ab if and only if it satisfies the Fermat test to base b.

(b) Show that if n is composite but not Carmichael, then n fails the Fermat test to base a for at least 50% of those a with $\gcd(a, n) = 1$ and $1 \le a \le n$.

(4) Factor $n = 140299$ using the Pollard ρ algorithm.

(5) Use the quadratic sieve to factor 498319.

(6) Factor the following numbers using a clever technique (meaning, not trial division) of your choosing:

(a) 8051	(c) 108781	(e) 1792939
(b) 10349	(d) 1337242	

(7) Use the Miller–Rabin test to show that 1729 is composite.

(8) If $2^p - 1$ is prime, then p must also be prime. However, the converse is false. Use the Miller–Rabin test to show that $2^{29} - 1$ is composite. (The smallest counterexample is $2^{11} - 1$. Do that one instead if you prefer to work with smaller numbers.)

(9) Use the AKS algorithm to show that 161 is composite.

(10) (a) The number N is the product of two n-digit primes p and q, where the first more than $n/2$ digits of p and q are the same. Explain how to factor N (and thus find p and q) given this knowledge.

(b) Factor 2130468073.

(11) We have seen various algorithms for attacking the discrete logarithm problem and integer factorization. Using big O notation, how many steps does it take for these algorithms to work, for a "typical" case? (For example, it might be that for certain special integers, it takes much longer to factor them than it does for most integers. If so, you can ignore them. Similarly, some of these algorithms involve some randomness, so you can assume that the random number generator isn't conspiring against you unreasonably.)

(12) What happens if we use a linear polynomial instead of a quadratic polynomial when applying the Pollard ρ algorithm? What is the running time, in terms of big O notation?

(13) Show that if p is prime and $a^2 \equiv 1 \pmod{p}$, then either $a \equiv 1 \pmod{p}$ or $a \equiv -1 \pmod{p}$. Find a counterexample if p is composite.

(14) Prove Lemma 13.6.

(15) Show that if there are n possible birthdays, all equally likely, then the number of people needed before the probability that two share a birthday is more than 50% is roughly $\sqrt{2n\log(2)}$. (Hint: Sbe fznyy k, r gb gur k vf nccebkvzngryl rdhny gb bar cyhf k.) Can you find an error term, i.e., can you write down a function $f(n)$ whose growth rate is smaller than that of $\sqrt{n}$ such that the number of people needed before the probability is 50% is $\sqrt{2n\log 2} + O(f(n))$?

(16) Show that if $6k+1$, $12k+1$, and $18k+1$ are all prime, then $(6k+1)(12k+1)(18k+1)$ is a Carmichael number. (This doesn't prove that there are infinitely many Carmichael numbers, because we do not yet know if it happens infinitely often that $6k+1$, $12k+1$, and $18k+1$ are all prime. We strongly believe this to be the case, however.)

(17) It is often important to be able to choose a random integer from 1 to N; this is easily done. However, sometimes it is desirable to choose a random *factored* integer from 1 to N. One can do this by first choosing a random integer and then factoring it, but factoring is hard. Show that the following algorithm produces a random integer from 1 to N, together with its prime factorization, quickly:

- Pick a random sequence $N \geq s_1 \geq s_2 \geq \cdots \geq s_t = 1$ by choosing s_1 uniformly in $\{1, \ldots, N\}$ and then choosing s_{i+1} uniformly in $\{1, \ldots, s_i\}$.
- Let r be the product of the *prime* s_i's.
- If $r \leq N$, output r with probability $\frac{r}{N}$. Otherwise, start over from the beginning.

In other words, show that the output of this algorithm is uniform on $\{1, \ldots, N\}$. (This algorithm comes from [Kal03].)

(18) Find an attack on the discrete logarithm problem based on the Pollard ρ algorithm.

(19) Let $\psi(x, y)$ denote the number of y-smooth numbers $\leq x$. Find the best upper and lower bounds you can on $\frac{\psi(x,y)}{x}$, the proportion of y-smooth numbers up to x. (You may find it convenient to write $y = x^{1/u}$ for some u.)

(20) Show that if $1 \leq u < 2$, then $\frac{\psi(x,x^{1/u})}{x} \approx 1 - \log(u)$. (You may use the Prime Number Theorem, which says that the number of primes $\leq x$ is roughly $\frac{x}{\log(x)}$.)

Chapter 14
Elliptic Curves

14.1 Rational Points on a Circle

The next cryptographic topic we will discuss will be elliptic curves. But before we get to the cryptography, we must define elliptic curves. And even before that, we will start with circles to get warmed up.

What are the rational points on the circle $x^2 + y^2 = 1$? That is, for which points (x, y), where both x and y are rational, is $x^2 + y^2 = 1$? Can we describe them nicely?

Here are some key observations (see Figure 14.1):

(1) Suppose we have two rational points (x_0, y_0) and (x_1, y_1) on the circle. Then, the slope of the line connecting them is $\frac{y_1 - y_0}{x_1 - x_0}$. The relevant feature for us is that this line has a *rational* slope.

(2) If (x_0, y_0) is a rational point, and t is a rational number, we can form a line through (x_0, y_0) with slope t. This line will intersect the curve at one more point, and that point will be rational as well. Why is that? Well, its x-coordinate will be rational because it is the solution of a quadratic equation with rational coefficients *that already has a rational root*. So the other root will be rational too. Now, the y-coordinate is rational because it is on the line $y = y_0 + t(x - x_0)$, so it's a sum and product of rational numbers. (If this sort of reasoning bothers you, you can work out the other root directly and see that it is indeed rational, as we do below.)

Hence, we can describe all the rational points on the circle by first finding one rational point, and then looking at all the lines with rational slopes through that point and finding the other intersection with the circle.

One convenient way of doing this is to start with the point $(-1, 0)$ on the circle. Suppose we look at the line with slope t through $(-1, 0)$. Where does it intersect the circle again? When we do the calculation out, we see that it intersects again at

$$\left(\frac{1-t^2}{1+t^2}, \frac{2t}{1+t^2}\right).$$

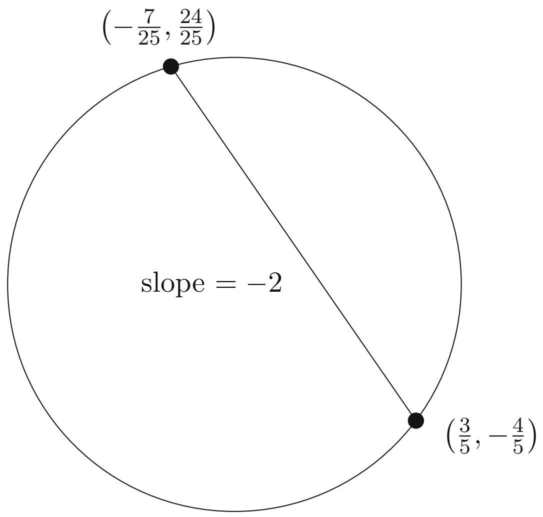

$\left(-\frac{7}{25}, \frac{24}{25}\right)$
slope $= -2$
$\left(\frac{3}{5}, -\frac{4}{5}\right)$

Figure 14.1 Two rational points on the circle and the line connecting them

Or, letting $t = \frac{m}{n}$, the point becomes

$$\left(\frac{m^2 - n^2}{m^2 + n^2}, \frac{2mn}{m^2 + n^2}\right).$$

Corollary 14.1 *Let (a, b, c) be a primitive Pythagorean triple, so that $a^2 + b^2 = c^2$ and $\gcd(a, b) = 1$. Then, for some integers (m, n) we have either $(a, b, c) = (m^2 - n^2, 2mn, m^2 + n^2)$ or $(a, b, c) = (2mn, m^2 - n^2, m^2 + n^2)$.*

This is because if (a, b, c) is any Pythagorean triple, then $(a/c, b/c)$ is a rational point on the circle. Conversely, if $(a/c, b/c)$ is any rational point on the circle, then $a^2 + b^2 = c^2$, so (a, b, c) is a Pythagorean triple. It will be in lowest terms if a/c and b/c are.

There are some restrictions we should put on m and n in order to get primitive Pythagorean triples: $\gcd(m, n)$ should be 1, and m and n should not both be odd. (What happens when they are both odd? Then $m^2 - n^2$, $2mn$, and $m^2 + n^2$ are all even, so this triple isn't primitive.)

What values of m and n give us interesting triples?

m	n	(a, b, c)
2	1	(3,4,5)
3	2	(5,12,13)
4	1	(8,15,17)
4	3	(7,24,25)
5	2	(20,21,29)
5	4	(9,40,41)

So, we easily recover many of our favorite primitive Pythagorean triples in this way.

14.2 Elliptic Curves

What are elliptic curves? First of all, what *aren't* they? They *aren't* ellipses. They're called that for historical reasons: they were first studied due to their appearance in the study of elliptic integrals, which arose out of trying to find the arc length of an ellipse.

Definition 14.2 An *elliptic curve* is a curve of the form

$$y^2 = x^3 + ax + b,$$

where a and b are such that $4a^3 + 27b^2 \neq 0$.

The $4a^3 + 27b^2 \neq 0$ bit says that the cubic polynomial $x^3 + ax + b$ has no repeated roots.

An elliptic curve looks like one of the following two curves, depending on whether the cubic equation has 1 or 3 real roots (see Figure 14.2).

Elliptic curves are wonderful things, and they are important in many areas of mathematics, including number theory, algebraic geometry, and topology. And they are also important in cryptography. The main reason that they are important is because there is an *abelian group* on an elliptic curve: the points of the elliptic curve will be the elements of the group. Thus, it will be possible to add points on an elliptic curve. We might first hope that we just add the x-coordinates and the y-coordinates, but this doesn't work: when we do that, we won't usually get another point on the elliptic curve.

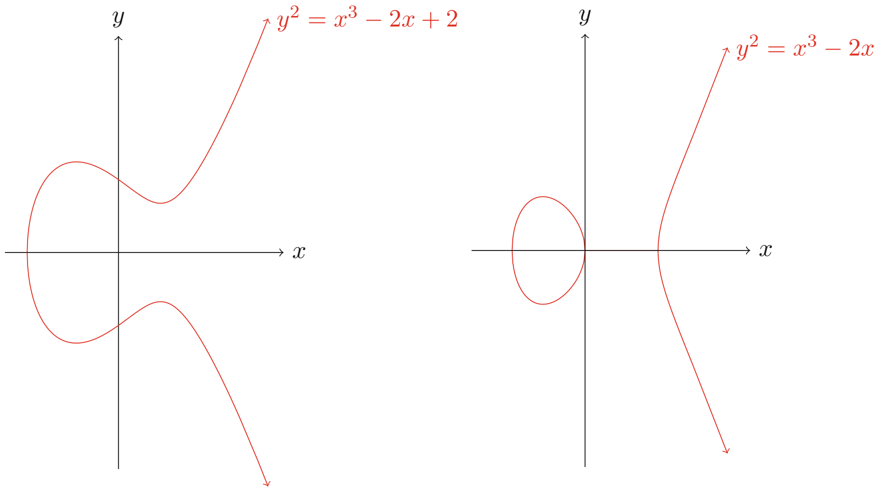


Figure 14.2 Elliptic curves of the form $y^2 = f(x)$. Top: f has one real root. Bottom: f has three real roots

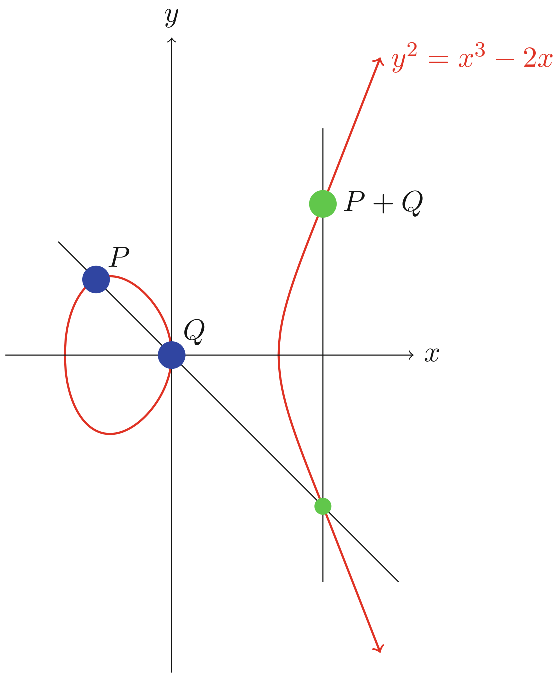


Figure 14.3 Elliptic curve addition

To define the group structure, we first have to add a point to the elliptic curve, a point at infinity. We will think of this point as lying on every vertical line and no non-vertical line. We will make the point at infinity into the identity of the group. The group operation is called addition. Take two points P and Q on an elliptic curve E. Draw the line connecting P and Q; this line intersects the elliptic curve at a third point. Then, we reflect this point across the x-axis; this point is $P + Q$. See Figure 14.3 for a picture.

We can write down a formula for addition as well: if we wish to add the points $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ on the elliptic curve $y^2 = x^3 + ax + b$, then we must first construct the line connecting them, which has slope

$$m = \frac{y_2 - y_1}{x_2 - x_1}$$

and intercept $c = y_1 - mx_1$, so that the equation of the line through P and Q is $y = mx + c$. The points of intersection of this line with the elliptic curve satisfy the equations

$$y^2 = x^3 + ax + b, \qquad y = mx + c.$$

Substituting $y = mx + c$ into the equation of the elliptic curve gives

$$(mx + c)^2 = x^3 + ax + b. \tag{14.1}$$

Now, we know two solutions for x, namely, x_1 and x_2. Let us say that the third point is (x_3, y_3). Moving everything in (14.1) to one side, we get

$$x^3 - m^2x^2 + \Box x + \Box = 0,$$

where I don't care what the $\Box$'s are. This is because we know that the interpretation of the opposite of the quadratic term as the sum of the roots: $m^2 = x_1 + x_2 + x_3$. Thus, we have $x_3 = m^2 - x_1 - x_2$. Furthermore, $y_3 = mx_3 + c$. Finally, $P + Q = (x_3, -y_3)$.

Example Consider the elliptic curve $E : y^2 = x^3 + x + 6$, and let $P = (2, 4)$ and $Q = (3, -6)$. These are both points on E. Let us compute $P + Q$. The line through P and Q is $y = -10x + 24$. Hence, with the above notation, we have $m = -10$ and $c = 24$. Their sum is $P + Q = (x_3, -y_3)$, where $x_3 = m^2 - x_1 - x_2 = 95$ and $y_3 = mx_3 + c = -10 \times 95 + 24 = -926$. Thus $P + Q = (95, 926)$.

There are some slight subtleties here: if we wish to double a point (i.e., add it to itself), then we make a line tangent to the curve at that point. Also, if two points are on the same vertical line, then that line won't contain a third point on the curve — except the point at infinity! So the sum of two points on the same vertical line (i.e., (x, y) and $(x, -y)$) is the point at infinity.

It is also important for us to write down the formula for doubling on the elliptic curve. If $P = (x_1, y_1)$ and we wish to compute $2P$, then we must compute the slope of the tangent line to E at P. It is a standard exercise in calculus to show that this slope is

$$m = \frac{3x_1^2 + a}{2y_1}.$$

The computation for $2P = (x_3, y_3)$ is now similar: we find the other point of intersection of the line $y = mx + c$ with $c = y_1 - mx_1$ and the elliptic curve $y^2 = x^3 + ax + b$. The formula is essentially the same, except instead of having $x_3 = m^2 - x_1 - x_2$, we have $x_3 = m^2 - 2x_1$.

Example On the same elliptic curve $y^2 = x^3 + x + 6$, let us compute $2P$, where $P = (2, 4)$ again. We have $m = \frac{13}{8}$ and $c = \frac{13}{4}$. Hence, we have

$$2P = \left(-\frac{87}{64}, \frac{747}{512}\right).$$

Note that we doubled a point with *integer* coordinates, but we only ended up with a point with *rational* coordinates. This is typical.

In fact, without going through all this stuff about writing down the line, we can give an explicit x-coordinate for $2P$: if $P = (x, y)$, then the x-coordinate of $2P$ is

$$\frac{x^4 - 2ax^2 - 8bx + a^2}{4(x^3 + ax + b)}. \tag{14.2}$$

The y-coordinate is then one of the two square roots of this number; it is not too difficult (by drawing a picture, say) to tell which square root we want.

Remark 14.3 If P is a point on an elliptic curve, then the x-coordinate of nP depends only on the x-coordinate of P. These functions of the x-coordinate, like the one in (14.2), are known as *Lattès maps*. These maps are very important in dynamical systems. See problem 10 in Chapter 15.

Most of the group properties are easy to check, but one of them is hard, and that is associativity. One, rather uncivilized, way of checking this is to use the formula for what $P + Q$ is in terms of the coordinates of P and Q, and then go through the rather horrific computations needed to show that it is indeed associative. Generally, people prefer to check associativity in fancier ways, and they will throw around terms like “the Riemann–Roch theorem” and possibly “complex tori” and “the Weierstraß $\wp$-function.” Those are wonderful topics that I encourage you to learn about, not necessarily immediately, but eventually. However, we’ll just accept that this operation is indeed associative and move on.

Remark 14.4 If we add two rational points on an elliptic curve, their sum is also a rational point, at least if we consider the point at infinity to be a rational point (which it fully deserves to be!). This is completely analogous to the case of the circle, where if we intersect a line with rational slope passing through a rational point of the circle, with the circle, then the other point of intersection is rational. (The rational points on a circle also form an abelian group, but in a different way.) The proof is completely analogous. The conclusion is that the rational points on an elliptic curve form a group.

There is much that is known about the rational points on elliptic curves, as well as many open problems. We will return to this topic in Section 15.4.

14.3 Elliptic Curves Over Finite Fields

However, it is not elliptic curves over the rational numbers that will be of primary interest to us in studying cryptography (although they are very interesting when studying number theory). Instead, we’ll be interested in elliptic curves over finite fields! What does an elliptic curve over $\mathbb{F}_p$ mean? Well, it’s the same sort of thing as an elliptic curve over $\mathbb{R}$, but now x and y are in $\mathbb{F}_p$. So, if E is the curve $y^2 = x^3 + ax + b$, then we will write $E(\mathbb{F}_p)$ to mean all the pairs of numbers (x, y) with $x, y \in \mathbb{F}_p$ so that $y^2 \equiv x^3 + ax + b \pmod{p}$, together with the point at ∞. We will similarly refer to $E(\mathbb{Q})$, $E(\mathbb{R})$, $E(\mathbb{Z})$, and other such things, to refer to the solutions of $y^2 = x^3 + ax + b$ such that x and y are in $\mathbb{Q}$, $\mathbb{R}$, or $\mathbb{Z}$, together with the point at ∞.

There is one important thing to watch out for here: if E is an elliptic curve such that a and b are in $\mathbb{Z}$, then we don't *necessarily* get an elliptic curve over $\mathbb{F}_p$ by considering the solutions modulo p. Recall that in order for $y^2 = x^3 + ax + b$ to be an elliptic curve, we required that $4a^3 + 27b^2 \neq 0$. That's still true over $\mathbb{F}_p$, but now equality in $\mathbb{Q}$ is replaced by equality in $\mathbb{F}_p$, also known as congruence: $4a^3 + 27b^2 \not\equiv 0 \pmod{p}$. If $4a^3 + 27b^2 \equiv 0 \pmod{p}$, then we do *not* get an elliptic curve over $\mathbb{F}_p$, but rather a different sort of beast. Fortunately, it turns out that if E is an elliptic curve with coefficients in $\mathbb{Z}$, then there are only finitely many of these "bad" primes[1] where $4a^3 + 27b^2 \equiv 0 \pmod{p}$, since these primes are simply the primes dividing $4a^3 + 27b^2$.

It probably isn't clear in what sense a bunch of points over $\mathbb{F}_p$ (and a point at ∞) deserves to be called a *curve*, since there isn't really any geometry going on there, but we'll call it one by analogy with the situation over the real numbers. (There is a much more general definition of a curve that makes an elliptic curve over $\mathbb{F}_p$ into an honest curve, but that's beyond the scope of this book.)

When we looked at elliptic curves over $\mathbb{R}$ or over $\mathbb{Q}$, we saw that we could add points using a geometric construction. This geometric construction involving lines no longer makes sense. However, we also wrote down a *formula* for adding them, and that formula still makes perfect sense over $\mathbb{F}_p$. Now, we don't have any geometric interpretation of adding points on the elliptic curve, but the formulae we worked out before still work, so we can still add points on an elliptic curve over $\mathbb{F}_p$. The points still form a group.

Example Let E be the elliptic curve $y^2 = x^3 + x + 1$. We will find all the points of $E(\mathbb{F}_5)$. They are the following:

$$\infty, (0, 1), (0, 4), (2, 1), (2, 4), (3, 1), (3, 4), (4, 2), (4, 3).$$

For example, $3^2 \equiv 4^3 + 4 + 1 \pmod{5}$, so $(4, 3) \in E(\mathbb{F}_5)$.

For large finite fields, it might be a pain to enumerate the points, so we can do this using Sage:

```
E = EllipticCurve(GF(5),[1,1])
E.points()
```

The `GF(5)` means $\mathbb{F}_5$. (GF stands for "Galois field," which is another name for a finite field, although somewhat dated.) To get Sage to work with the elliptic curve $y^2 = x^3 + ax + b$, replace the [1,1] with $[a, b]$. The command `E.points()` gives us all the points. It puts an extra :1 at the end, for reasons that we won't get into now. The lone exception is the point (0:1:0), which is the point at infinity.

Sage is also happy to add points for us, and we really don't want to have to do that by hand! Suppose we want to add the points $(0, 1)$ and $(2, 1)$. To do this we give them names and then tell Sage to add:

[1] Officially known as "primes of bad reduction."

```
P = E(0,1)
Q = E(2,1)
P+Q
```

and Sage tells us that the answer is (3:4:1), or (3, 4). We can also type `2*P` to get it to add P to itself.

14.4 Point Counting on Elliptic Curves

Can we estimate by pure thought how many points $E(\mathbb{F}_p)$ should contain? Here is the idea: an elliptic curve is the set of points of the form $y^2 \equiv f(x) \pmod{p}$, together with one more point at infinity. Now, we have two points on E with a given x-coordinate if $f(x)$ is a nonzero square modulo p, one if $f(x) = 0$, and none if $f(x)$ is not a square modulo p. So, how often is $f(x)$ a square modulo p?

Well, $f(x) \pmod{p}$ is basically random, so we can ask how often a random element of $\mathbb{F}_p$ is a square modulo p. Disregarding 0 for now, an element of $\mathbb{F}_p^\times$ either has two square roots, or it has none. Since there are $p - 1$ elements of $\mathbb{F}_p^\times$ to square, and their squares come in pairs since a and $-a$ have the same square modulo p, half of the elements of $\mathbb{F}_p^\times$ are squares, and half of them aren't.

Thus, since we think that $f(x) \pmod{p}$ is basically random, we think it should be a square something like half the time, so with probability $1/2$ there will be two points on E with a given x-coordinate, and with probability $1/2$ there will be none. So, the average number of points with a given x-coordinate is 1. (We ignored the ones with $f(x) \equiv 0$, but they contribute 1 as well, so we're still safe!)

Thus, we expect around p points in $E(\mathbb{F}_p)$, together with the point at infinity, for a total of $p + 1$. (See problem 14 for more details of this calculation.) This analysis is indeed correct. There is a more precise version, called the Hasse–Weil bound:

Theorem 14.5 (Hasse–Weil bound, [Has36a, Has36b, Has36c]). *Let E be an elliptic curve over $\mathbb{F}_p$. Let $\#E(\mathbb{F}_p)$ be the number of points in $E(\mathbb{F}_p)$. Then*

$$|\#E(\mathbb{F}_p) - (p + 1)| \leq 2\sqrt{p}.$$

In other words, $p + 1$ isn't too far off.

14.5 A Smattering of Group Theory

Since we have only been introducing group theory on a "need-to-know" basis, I have to give you a collection of facts about group theory that are needed for the study of elliptic curves.

Definition 14.6 Let G be a finite group (i.e., one with finitely many elements) with identity element e, and let $g \in G$ be an element. Then, the *order* of g is the smallest $n > 0$ so that $g^n = e$.

The following theorem about orders of elements, due to Lagrange, is probably the first interesting theorem in group theory:

Theorem 14.7 (Lagrange). *The order n of an element g divides the number of elements in the group G.*

Proof The idea is to partition the group G up into several sets, each of size n, so that every element of G is in exactly one of them. To do this, start with some $a \in G$, and consider the set $H_a = \{a, ag, ag^2, \dots, ag^{n-1}\}$. We claim that all these elements are different, so that H_a consists of n elements of G. To see this, suppose that two of them were equal, say $ag^i = ag^j$, where $i < j$. Rearranging a bit, this is equivalent to $g^{j-i} = e$. But n is the order of g, i.e., the *least* positive integer n such that $g^n = e$, and $0 < j - i < n$. Thus, this cannot be, and indeed all the elements ag^i in H_a are distinct.

Now, every element $b \in G$ is contained in some such set H_a, since $b \in H_b$. Now, suppose that two of these sets, say H_a and H_b, have an element $c = ag^i = bg^j$ in common. Then, we claim that $H_a = H_b$. The reason is that if we have an arbitrary element $ag^k \in H_a$, then we can write this same element as

$$cg^{k-i} = bg^j g^{k-i} = bg^{j+k-i},$$

where we take the exponent $j + k - i$ modulo n if necessary. Thus, the number of such distinct sets H_a is the size of G divided by n, which must be an integer. Hence, n divides the number of elements in G. ■

As a consequence, if a group has a prime number p of elements, then it is a cyclic group, and every element other than the identity has order p.

Definition 14.8 If in an abelian group G, $n \cdot g = \underbrace{g + g + \cdots + g}_{n} = e$, then we say that g is an *n-torsion element* (or n-torsion *point* if G consists of points on an elliptic curve). We write $G[n]$ for the set of n-torsion elements in G.

Remark 14.9 In an abelian group, $G[n]$ is actually a group: if we add two n-torsion elements, we get another n-torsion element, and similarly with inverses.

Another thing that will be useful is the notion of a direct product. Let G and H be two groups. Then, we can form a new group $G \times H$ out of them, as follows: the elements of $G \times H$ are ordered pairs (g, h), where $g \in G$ and $h \in H$. We multiply two elements of $G \times H$ using the rule

$$(g_1, h_1)(g_2, h_2) = (g_1 g_2, h_1 h_2).$$

This multiplication turns $G \times H$ into a group. (See problem 11.)

We can iterate this direct product construction with three or more groups: if $G_1, \ldots, G_n$ are groups, then we can form the direct product $G_1 \times \cdots \times G_n$. (This also works for infinite products, although in this case there are two different sorts of "infinite products" we can form. The more obvious of the two is called the direct product, and the slightly more subtle one is called the direct sum.)

14.6 The Group Structure on $E(\mathbb{F}_p)$

It turns out that $E(\mathbb{F}_p)$ can always be generated by at most two points; this is specific to the case of a finite field and is not always true over other fields like $\mathbb{Q}$. Suppose that $P \in E(\mathbb{F}_p)$, and let $\langle P \rangle$ denote the cyclic group consisting of all multiples of P in $E(\mathbb{F}_p)$, i.e., the group generated by P. Then, the fact that $E(\mathbb{F}_p)$ can be generated by (at most) two elements can be translated as saying that $E(\mathbb{F}_p) = \langle P \rangle \times \langle Q \rangle$ for some $P, Q \in E(\mathbb{F}_p)$. (This direct product decomposition is something special about *abelian* groups, and it is very far from being true for nonabelian groups.)

Sometimes $E(\mathbb{F}_p)$ is a cyclic group, meaning that it can be generated by a single element, but this does not always happen. However, when it does happen that $E(\mathbb{F}_p)$ is cyclic, then E *might* be a good elliptic curve for certain aspects of cryptography, such as the elliptic curve versions of Diffie–Hellman and ElGamal. (However, not always: if $\#E(\mathbb{F}_p) = p$, then $E(\mathbb{F}_p)$ is definitely cyclic, but E is terrible for cryptography, as shown in [Sma99]. Curves E with $\#E(\mathbb{F}_p) = p$ are called *anomalous* curves.)

14.7 Elliptic Curve Diffie–Hellman

Naturally, we're interested in elliptic curves for their applications to cryptography. Cryptosystems involving discrete logarithms over $\mathbb{F}_p$ and factoring are pretty good, but there are some that are even more secure, coming from elliptic curves.

For example, there is an elliptic curve version of Diffie–Hellman, first introduced independently by Neal Koblitz and Victor Miller in the two papers [Kob87, Mil86]. More generally, whenever we have a finite cyclic group G, we can create a version of Diffie–Hellman. Its efficacy as a cryptosystem depends on several factors, most notably whether it is easy to solve the discrete logarithm problem in G. Another important factor is the computational complexity of performing the group operation in G.

The elliptic curve version of Diffie–Hellman works pretty much the same way as the standard Diffie–Hellman, but instead of picking a generator g of $\mathbb{F}_p^\times$, we start with a point on an elliptic curve. More precisely: Alice and Bob pick a prime p, an elliptic curve E over $\mathbb{F}_p$, and a point P on E. These three things are public knowledge. Then, Alice picks some number a and computes aP, and Bob picks some number b and computes bP. They send these points to each other. Then, Alice computes

$a(bP)$, and Bob computes $b(aP)$. These are the same point: abP. It is hard for Eve to reconstruct abP from knowing P, aP, and bP.

This is analogous to the version of Diffie–Hellman we saw before: just think of P as being like g.

Similarly, there is an elliptic curve version of ElGamal. It starts the same way as elliptic curve Diffie–Hellman: Alice picks a prime p, an elliptic curve E over $\mathbb{F}_p$, and a point P on E. She also picks a number a and computes aP. The prime p, the elliptic curve E, and the points P and aP are all public knowledge. Now, to send a message m, which is a point on E, Bob picks a random b and computes bP and $m + abP$, and those two together are his ciphertext. To decrypt the message, Eve computes $(m + abP) - a(bP) = m$ and recovers the message.

Let's try it out!

Example Let E be the elliptic curve $y^2 = x^3 + 5x + 6$ over $\mathbb{F}_{191}$. This curve has 196 points, and the group is a cyclic group generated by a single point: $P = (178, 185)$. You can find this out using Sage by typing

```
E = EllipticCurve(GF(191),[5,6])
E.gens()
```

We will choose P as our point on E. Suppose we're Bob, and we don't know what Alice has chosen, but she tells us that $aP = (125, 141)$. Now, we would like to give Alice the message $(4, 89)$. We randomly choose a b, and let us say that we randomly choose $b = 132$, so that $bP = (161, 178)$. We compute $132(aP) + m$ and get $(4, 102)$. So, our ciphertext consists of the points $(161, 178)$ and $(4, 102)$.

In order for Alice to decrypt it, she must remember what a is, and $a = 67$. She can recover abP, since it is $67(bP) = (64, 6)$. Now, she can recover the message, since it is $(4, 102) - (64, 6) = (4, 89)$, which is correct.

There is at least one serious point here: how does Bob encode something as a point on E? First, he encodes it as a number modulo p, and then he wants to encode it as a point with that x-coordinate. But there's a problem! What if that number *isn't* the x-coordinate of any point on E? Then, he's stuck! So, instead, what he does is to associate it to a range of many numbers, perhaps 100 of them, and then he finds one of them that's the x-coordinate of a point on E. Since each point has a 50% chance of being an x-coordinate of a point on E, he's almost certain to find at least one that works!

Example Let's see an example of how to perform encryption with elliptic curve ElGamal using a message consisting of text rather than numbers or points on an elliptic curve. The first thing that Alice and Bob must agree on, over a public channel if desired, is the protocol for converting text to numbers. Let us suppose that they elect to use the very simple method of encoding `A` as 1, `B` as 2, and so forth. The next thing that Alice must do is to choose a prime p and an elliptic curve over $\mathbb{F}_p$. She takes $p = 270001$ and the elliptic curve

$$E : y^2 = x^3 + 154302x + 251808.$$

The group of points over $\mathbb{F}_p$ is cyclic of order 269893, generated by the point $P =$ (205152, 143044). Alice also chooses a number a as her private key. I won't tell you what it is, but $Q = aP =$ (73373, 78463). Her public key then consists of the data of the prime p, the elliptic curve E, the generator P, and the point Q. These become public information.

Now, Bob wishes to send Alice a message. Let us suppose his message is `hello`. In numbers, this is 8, 5, 12, 12, 15. First, he must convert each of these letters/numbers to points on E. He has enough space that he can afford to assign to each letter a range of size 10000: the letter `h` (number 8) corresponds to the range 80000–89999, for instance. He must find a point on E with x-coordinate in the range 80000–89999, so there are tons of choices available to him. Indeed, 80000 works. He does the same thing for the other letters, and he finds that his message corresponds to the following sequence of points on E:

$$(80000, 80609),\ (50001, 30531),\ (120004, 83763),\\ (120004, 83763),\ (150002, 18516).$$

For each one, Bob must choose a random b. It is important to use a different value of b each time, so that the two `l`'s get encoded differently, so that Eve cannot tell that they are encryptions of the same letter, and thus get a hint of the message. He chooses the following five values of b:

$$14415,\ 61572,\ 9190,\ 151600,\ 78033.$$

His encryption for the first letter consists of the two points bP and $bQ + M$. He computes these and gets

$$bP = (232531, 6816), \qquad bQ + M = (173346, 268989).$$

He encrypts the other four letters in the same way, so that his entire message consists of the following ten points:

$$\begin{array}{rr} (232531, 6816) & (173346, 268989) \\ (15538, 87103) & (191306, 84979) \\ (202639, 185807) & (99364, 236398) \\ (165778, 179250) & (144259, 248915) \\ (103945, 109461) & (238572, 200292). \end{array}$$

Each line corresponds to a single character in the message. This seems rather inefficient, but of course the protocol for turning messages into points can be done in a more concise manner, and it's probably a good idea to do some blocking as we discussed for RSA, even though ElGamal doesn't suffer from the same flaw with respect to unblocked messages that RSA does.

14.8 Why Elliptic Curves?

Why would we use elliptic curve Diffie–Hellman over normal Diffie–Hellman? One reason is the existence of specialized attacks on the discrete logarithm problem over $\mathbb{F}_p^\times$ that do not work over other abelian groups like elliptic curves. The most famous of these attacks is the index calculus that we discussed in Section 11.7. The existence of such attacks means that in order for normal Diffie–Hellman to be secure, we need to use a very large prime p. But since these methods don't work for elliptic curve Diffie–Hellman, we can get away with using a smaller prime p in the elliptic curve case, which is more efficient.

For cryptographic purposes, it would be great to have an abelian group that "admits no tricks." That is, there is nothing special about it that makes it easier to deal with than some other abelian groups. To take an extreme example, the cyclic group $\mathbb{Z}/n\mathbb{Z}$ under addition is bad cryptographically because if we know g and ag, then it is easy to compute a: we can do this by *division modulo* n. This operation is outside the realm of the group $\mathbb{Z}/n\mathbb{Z}$, but nonetheless an attacker can take advantage of its existence: as the adage goes, all is fair in love, war, and cryptanalysis!

The multiplicative groups $(\mathbb{Z}/p\mathbb{Z})^\times$ are much more secure than $\mathbb{Z}/n\mathbb{Z}$, but they too suffer from specialized attacks that make solving the discrete logarithm problem, while not really easy, at least doable for a larger range of primes p than one might expect. Elliptic curves, at least well-chosen ones, appear to be rather immune to specialized attacks, but are still not too hard to compute with.

In fact, there are methods of "converting" attacks on the discrete logarithm problem for $\mathbb{F}_q^\times$ for a prime *power* $q = p^r$ into attacks on the discrete logarithm problem for elliptic curves over $\mathbb{F}_p$; see, for instance, [Die11]. However, for well-chosen elliptic curves, these attacks drastically increase the size of the cyclic group by forcing r to be very large, making them impractical and certainly much slower than attacks against $\mathbb{F}_p^\times$ of similar key size.

There are other advantages to elliptic curve Diffie–Hellman as well, due to the fact that elliptic curves carry such a rich structure. For example, there is a *three-way* key exchange with elliptic curve Diffie–Hellman [Jou04]. That is, Alice, Bob, and Charlie want to come up with a common key. They can do this using the classical Diffie–Hellman key exchange, if they have several rounds of exchanging information. However, with elliptic curves, they can come up with a common key with only a single round of information exchange. This three-way key exchange relies on a gadget called the Weil pairing on elliptic curves, that is beyond the scope of this book. We do not know of a four-way one-round key exchange using elliptic curves. However, Sanjam Garg, Craig Gentry, and Shai Halevi have recently proposed a one-round n-way key exchange using entirely different methods [GGH13].

14.9 Problems

(1) Find a rational point on the elliptic curve $y^2 = x^3 + x + 3$. How many can you find? How many of them have integer coordinates?

(2) Show that the cubic polynomial $x^3 + ax + b$ has a double zero (i.e., it factors as $(x - c)^2(x - d)$ or $(x - c)^3$) if and only if $4a^3 + 27b^2 = 0$.

(3) We parametrized the rational points on the circle $x^2 + y^2 = 1$ as $\left(\frac{1-t^2}{1+t^2}, \frac{2t}{1+t^2}\right)$, together with the point $(-1, 0)$. Parametrize the rational points on the hyperbola $x^2 - 2y^2 = 1$ in a similar way, involving some parameter t.

(4) Use your answer to the previous problem to find interesting integer solutions to $x^2 - 2y^2 = z^2$.

(5) Consider a conic section $C : ax^2 + bxy + cy^2 + dx + ey + f = 0$, over the field $\mathbb{F}_p$. What is the maximum number of points it can have in $\mathbb{F}_p$? What if C is nondegenerate (i.e., $b^2 - 4ac \neq 0$)?

(6) Let E be an elliptic curve over a field F. Let P and Q be a point on E. What are the possible values for the number of points Q on E such that $2Q = P$?

(7) The curves $y^2 = x^3$ and $y^2 = (x + 2)(x - 1)^2$ are not elliptic curves, because they satisfy $4a^3 + 27b^2 = 0$.

  (a) Show that we can parametrize all their rational points in a convenient manner.
  (b) What happens if we try to perform the group law on these curves? Are these operations related to other more familiar groups?

(8) Let E be the elliptic curve $y^2 = x^3 + 2x + 4$. Write down all the points of $E(\mathbb{F}_5)$. Do the same for $E(\mathbb{F}_7)$ and $E(\mathbb{F}_{11})$.

(9) Alice and Bob decide to use elliptic curve Diffie–Hellman with the elliptic curve $y^2 = x^3 + 5x + 17$ over $\mathbb{F}_{89}$ with $P = (68, 68)$. Alice chooses $a = 56$, and Bob chooses $b = 32$. What is their shared key?

(10) Show that if G is an abelian group and n is a positive integer, then $G[n]$ is a group.

(11) Check that our definition of multiplication on $G \times H$ actually turns $G \times H$ into a group.

(12) Suppose that P is a 2-torsion point on an elliptic curve E, and that P is not the point at infinity. Show that the y-coordinate of P is 0.

(13) Show that P is a 3-torsion point on an elliptic curve E if and only if either $P = \infty$ or P is an inflection point of E. (An *inflection point* of a curve is one at which the second derivative is 0. Equivalently, a point is an inflection point if and only if the multiplicity of intersection between the curve and the tangent line is > 2.)

(14) (a) For how many $a \in \mathbb{F}_p$ is a a square? (That is, for how many $a \in \mathbb{F}_p$ is there some b for which $a \equiv b^2 \pmod{p}$?)
  (b) Suppose $f(x)$ is a "random" function from $\mathbb{F}_p$ to itself. How many solutions do you expect to have to the equation $y^2 \equiv f(x) \pmod{p}$? (Your answer should be an estimate, which depends on p but not on f. The exact number should depend on f, so your answer won't be exactly right.)

(15) Let E be the elliptic curve $y^2 = x^3 + ax + b$ defined over a field F in which $6 \neq 0$, and define its *j-invariant* to be

$$j(E) = 1728 \times \frac{4a^3}{4a^3 + 27b^2}.$$

Show that two elliptic curves $E : y^2 = x^3 + ax + b$ and $E' : y^2 = x^3 + a'x + b'$ have the same j-invariant if and only if there is some number c such that $a' = c^2a$ and $b' = c^3b$. (In other words, the elliptic curves are the same up to isomorphism, or, roughly speaking, scaling.) Note that c need not lie in F; we may first have to extract square roots or cube roots.

(16) Show that, for every $j \in F$, where F is a field with $6 \neq 0$, there is an elliptic curve defined over F with j-invariant j. Hence, the j-invariant parametrizes elliptic curves up to isomorphism, in the sense that the j-invariant gives a bijection between F and the set of elliptic curves over F up to isomorphism.

Chapter 15
The Versatility of Elliptic Curves

Elliptic curves come up everywhere in mathematics. We will now look at some of their uses in unexpected places. In this chapter, we will discuss some of these uses for elliptic curves. In particular, we will discuss a factorization algorithm based on elliptic curves, as well as some applications of elliptic curves over $\mathbb{Q}$ and $\mathbb{Z}$ to Diophantine equations. Before we can get to the elliptic curve factorization algorithm, we start with a non-elliptic curve factorization algorithm, that carries some of the same ideas as the elliptic curve method. This is the Pollard $p - 1$ algorithm.

15.1 Pollard's $p - 1$ Factorization

Let's suppose we wish to factor a number n, and that p is a prime factor of n (unknown to us). We know from Fermat's Little Theorem that if we pick any a not a multiple of p, then $a^{p-1} \equiv 1 \pmod{p}$, and more generally that for any integer k, $a^{k(p-1)} \equiv 1 \pmod{p}$.

So, here's a possible method for finding a factor of n: take some random number a and compute $s = a^m - 1$ for some multiple m of $p - 1$. Since s is a multiple of p, $p \mid \gcd(s, n)$, so unless we are unlucky and s is already a multiple of n, $\gcd(s, n)$ is a proper divisor of n, and we have found a factorization. In the case that $n = pq$ is the product of two primes, we have the full factorization.

But so far, we do not know how to find an m such that m is a multiple of $p - 1$. In general, this is quite hard. However, if $p - 1$ is B-smooth for some relatively small B, then it is not so hard.

Suppose we know that $p - 1$ is B-smooth, for some B. Then, we can write

$$p - 1 = 2^{e_2} 3^{e_3} \cdots q^{e_q},$$

S. Rubinstein-Salzedo, *Cryptography,* Springer Undergraduate Mathematics Series, https://doi.org/10.1007/978-3-319-94818-8_15

where q is the largest prime $\leq B$, and for each prime $\ell \leq B$, we have $e_\ell \leq b_\ell := \lfloor \log_\ell(p-1) \rfloor$. So, if we let

$$m = 2^{b_2} 3^{b_3} \cdots q^{b_q},$$

then m is *definitely* a multiple of $p-1$, so we can let $s = a^m - 1$ above, and this works.

Except that we *still* don't know p, so it is hard to compute the b_ℓ's. But that's okay; we just increase them. Instead, let $c_\ell := \lfloor \log_\ell n \rfloor$. Clearly, $c_\ell \geq b_\ell$, so we can safely choose

$$m = 2^{c_2} 3^{c_3} \cdots q^{c_q}.$$

This is a factorization algorithm known as the Pollard $p-1$ algorithm [Pol74]. In order to implement it effectively, we need to be slightly careful about how we perform the computations. Note that we do not need to know s exactly, and its exact computation is very slow as s is extremely large. Instead, we only need to know $s \pmod{n}$. Thus, when we compute powers of a, we may do so modulo n.

The above algorithm only works when $p-1$ is B-smooth, and in practice, we do not know a smoothness bound for $p-1$ in advance. Thus, it might make sense not to choose a smoothness bound in advance, but rather to create some iterative version that allows for an increase in smoothness as we go on.

Note also that, since we are computing numbers like $a^{2^{c_2} 3^{c_3} \cdots q^{c_q}} \pmod{n}$, we can compute $a_2 = a^{2^{c_2}} \pmod{n}$, then $a_3 = a_2^{3^{c_3}} \pmod{n}$, then $a_5 = a_3^{5^{c_5}} \pmod{n}$, and so forth.

Based on the above, here is a possible implementation of the Pollard $p-1$ algorithm:

(1) Pick some integer a, and don't do anything stupid like taking $a = 1$.
(2) For each prime q, let $c_q = \lfloor \log_q n \rfloor$.
(3) Let q_i denote the ith prime. Let $a_2 = a^{2^{c_2}} \pmod{n}$.
(4) Compute $g = \gcd(a_2 - 1, n)$. If $1 < g < n$, then g is a factor of n, and we have factored n. Otherwise, continue.
(5) Suppose we have computed $a_{q_{i-1}}$, and we have not yet found a factor of n. Then, let

$$a_{q_i} = a_{q_{i-1}}^{q_i^{c_{q_i}}} \pmod{n},$$

and compute $g = \gcd(a_{q_i} - 1, n)$.
(6) If $g = 1$, return to step (5). If $g = n$, then go back to step (1) and choose a different value of a. If $1 < g < n$, then we have found a factor of n, and we're done.

Here is a Sage implementation of the above:

```
def pollardfactor(n,a):
    q = 2
    while True:
        g = gcd(a-1,n)
        if g>1:
            return g
        cq = floor(log(n,q))
        a = power_mod(a,q^cq,n)
        print q, a
        q = next_prime(q)
```

Example Let $n = 57811239209$. This number has a prime factor p such that $p-1$ only has tiny prime factors. Thus, it is extremely quick to factor using the `pollardfactor` function above. Indeed, here is the output of running `pollardfactor(n,3)`:

```
2 24013905498
3 55425308742
5 38612217553
8101
```

Thus, $8101 = 2^2 \times 3^4 \times 5^2 + 1$ is a factor of n. The other factor is $q = 7139309$, but that didn't matter. Indeed, $q-1$ is not especially smooth: its largest prime factor is 6089.

This algorithm works very well when n has a prime factor p for which $p-1$ is B-smooth for some small B because if there are $\pi(B)$ primes up to B, then the algorithm only requires $\pi(B)$ passes through the loop. Again, we see the value of *safe primes* in cryptography: if for every prime factor p of n, $\frac{p-1}{2}$ is prime, then the Pollard $p-1$ algorithm is very slow. More generally, if $p-1$ has *some* large prime factor, then this algorithm is still slow.

What can we do about this? Why is $p-1$ special when factoring?

In fact, it isn't really. There is a modification of the Pollard $p-1$ algorithm, known as the Williams $p+1$ algorithm, which is fast if $p+1$ is smooth [Wil82]. More generally, Bach and Shallit [BS89] produced an algorithm that factors n quickly if $\Phi_k(p)$ is smooth for some cyclotomic polynomial Φ_k; the first several cyclotomic polynomials are

$$\begin{aligned}
\Phi_1(n) &= n - 1\\
\Phi_2(n) &= n + 1\\
\Phi_3(n) &= n^2 + n + 1\\
\Phi_4(n) &= n^2 + 1\\
\Phi_5(n) &= n^4 + n^3 + n^2 + n + 1\\
\Phi_6(n) &= n^2 - n + 1.
\end{aligned}$$

Ideally, we would like to be able to replace $p - 1$ by other numbers close to p that may be smooth. We can do so using elliptic curves, but before we can describe Lenstra's elliptic curve factorization, we need to talk about elliptic curves over $\mathbb{Z}/n\mathbb{Z}$, where n is composite.

15.2 Elliptic Curves Over $\mathbb{Z}/n\mathbb{Z}$

Given an elliptic curve $y^2 = x^3 + ax + b$ in short Weierstraß form, we can consider its solutions modulo n for any n. (For technical reasons, we usually want n not to be divisible by 2 or 3, since elliptic curves "should" look a bit different in those cases. In these cases, we can only write an elliptic curve in the form $y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$. See if you can understand why I named the coefficients in this peculiar manner.) This is "usually" an elliptic curve over $\mathbb{Z}/n\mathbb{Z}$, but remember that we need to be a bit careful: in order for $y^2 = x^3 + ax + b$ to be an elliptic curve over a field like $\mathbb{F}_p$, we need $4a^3 + 27b^2 \neq 0$ in $\mathbb{F}_p$ (i.e., $4a^3 + 27b^2 \not\equiv 0 \pmod{p}$), since if $4a^3 + 27b^2 = 0$, then the cubic $x^3 + ax + b$ has a double or triple root.

Over $\mathbb{Z}/n\mathbb{Z}$, several things are more subtle, and we will discuss the correct generalizations shortly; as we will soon see, we need something a bit stronger than just that $4a^3 + 27b^2 \neq 0$ in $\mathbb{Z}/n\mathbb{Z}$. Before we describe the precise criterion needed for $y^2 = x^3 + ax + b$ to be an elliptic curve, let us take a look at an example. If E is an elliptic curve with coefficients in $\mathbb{Z}/n\mathbb{Z}$, we can consider its points in $\mathbb{Z}/n\mathbb{Z}$, which we denote $E(\mathbb{Z}/n\mathbb{Z})$.

Example Consider the elliptic curve $E : y^2 = x^3 + 8x + 13$ over $\mathbb{Z}/35\mathbb{Z}$. One point on this elliptic curve is $(8, 22)$, since $22^2 \equiv 8^3 + 8 \times 8 + 13 \pmod{35}$. (Both sides are $29 \pmod{35}$). A few others are $(23, 3)$ and $(24, 13)$.

We also need to be careful about what happens at ∞. The points at ∞ are a convention needed to make the group law (and everything else) on the elliptic curve work out properly. So, it is not so easy to "logic" our way to the correct notion of point(s) at ∞ on $E(\mathbb{Z}/n\mathbb{Z})$ from first principles. (One can do so given more sophisticated understanding of the nature of elliptic curves, using algebraic geometry, but that is beyond the scope of this book.)

Instead, let us understand what happens with the non-infinite points. Let us consider the point $P = (8, 22)$ on the elliptic curve $E : y^2 = x^3 + 8x + 13$ over $\mathbb{Z}/35\mathbb{Z}$. Instead of considering E as an elliptic curve modulo 35, we can consider it as an elliptic curve over $\mathbb{F}_p$, for any prime factor p of 35. In this case, that's 5 and 7. Over $\mathbb{F}_5$, P reduces to $P_5 = (3, 2)$, and over $\mathbb{F}_7$, P reduces to $P_7 = (1, 1)$. Furthermore, if we start with P_5 over $\mathbb{F}_5$ and P_7 over $\mathbb{F}_7$, then using the Chinese Remainder Theorem on the x- and y-coordinates, we uniquely reconstruct the point P over $\mathbb{Z}/35\mathbb{Z}$.

What this means is that, among the *finite* points, there is a bijection between points on $E(\mathbb{Z}/35\mathbb{Z})$ and pairs of points (P_5, P_7), where $P_5 \in E(\mathbb{F}_5)$ and $P_7 \in E(\mathbb{F}_7)$ are *finite* points.

Now, it's clear what the most reasonable thing to do with the infinite points should be: we extend the bijection to *include* the infinite points. Thus, the points P on $E(\mathbb{Z}/35\mathbb{Z})$ should be in bijection with pairs of points (P_5, P_7), which may or may not be infinite points.

What this means is that we have *many* infinite points: one for each finite point P_5 of $E(\mathbb{F}_5)$ (corresponding to (P_5, ∞)), one for each finite point P_7 of $E(\mathbb{F}_7)$ (corresponding to (∞, P_7)), and one extra point which corresponds to (∞, ∞). We thus have $\#E(\mathbb{Z}/35\mathbb{Z}) = \#E(\mathbb{F}_5) \times \#E(\mathbb{F}_7)$.

Now, what's the deal with the discriminant $4a^3 + 27b^2$? What is the analog of "being nonzero" over $\mathbb{Z}/n\mathbb{Z}$? It *might* be that we require that $4a^3 + 27b^2 \not\equiv 0 \pmod{n}$. But "having a double root" is not so clear in the composite n case. For example, a degree-d polynomial can have more than d roots in $\mathbb{Z}/n\mathbb{Z}$. So, this suggests that the most obvious extension isn't exactly right.

Above, we had $E(\mathbb{Z}/35\mathbb{Z}) = E(\mathbb{F}_5) \times E(\mathbb{F}_7)$ (at least as far as the number of points goes, but in fact also for group laws, and everything else). So, in order to consider E to be an elliptic curve over $\mathbb{Z}/35\mathbb{Z}$, its reductions to $E(\mathbb{F}_5)$ and $E(\mathbb{F}_7)$ had better also be elliptic curves. This means that we need $4a^3 + 27b^2 \not\equiv 0 \pmod{5}$ and $4a^3 + 27b^2 \not\equiv 0 \pmod{7}$. Equivalently, we require that $\gcd(4a^3 + 27b^2, 35) = 1$. More generally, in order for $y^2 = x^3 + ax + b$ to be an elliptic curve over $\mathbb{Z}/n\mathbb{Z}$, we require that $\gcd(4a^3 + 27b^2, n) = 1$.

That's all we need to know about elliptic curves over $\mathbb{Z}/n\mathbb{Z}$ at the moment, so let us return to the matter of factorization and see how elliptic curves can be used to factor numbers.

15.3 Lenstra's Elliptic Curve Factorization

Recall that the Pollard $p - 1$ algorithm works when $p - 1$ is smooth for some prime factor p of n. Its modifications, the Williams algorithm and the Bach–Shallit algorithm, work when certain very specific other related numbers are smooth. But what do we do if, say, $p - 3$ is smooth? Is there a way to factor n in that case?

The real key to the Pollard $p - 1$ algorithm was that we have a group $\mathbb{F}_p^\times$ whose *order* is smooth. But there are other groups out there, where computations will have something to do with looking at something modulo p. A rich source of such groups is elliptic curves, and especially elliptic curves over $\mathbb{F}_p$. A very important theorem that we mentioned last chapter, and whose proof involves some algebraic geometry and is thus beyond the scope of this book, is the Hasse–Weil bound:

Theorem 15.1 (Hasse–Weil Bound, [Has36a, Has36b, Has36c]) *Let E be an elliptic curve over a finite field $\mathbb{F}_q$, and let $\#E(\mathbb{F}_q)$ denote the number of points on E. Then we have*

$$|q + 1 - \#E(\mathbb{F}_q)| \leq 2\sqrt{q}.$$

Thus, there are several choices for the number of points on an elliptic curve over $\mathbb{F}_p$. If we can find an elliptic curve E over $\mathbb{Z}/n\mathbb{Z}$ such that $\#E(\mathbb{F}_p)$ is smooth, then we will be able to factor n quickly. Since we don't know what p is, it will be hard to do this directly, so instead we'll try a bunch of elliptic curves until we get lucky. So, we do the following. Pick an elliptic curve E over $\mathbb{Z}/n\mathbb{Z}$, and find some point P on E. Choose some smoothness bound B, and compute m as in the Pollard $p-1$ factorization algorithm:

$$m = 2^{c_2}3^{c_3}5^{c_5}\cdots q^{c_q},$$

where q is the largest prime $\leq B$.

Now, we *attempt* to compute mP, i.e., $P + P + \cdots + P$ m times. However, we might fail in our computation of mP. What can go wrong? Let us remember how to double a point on an elliptic curve. To compute $2P$, we find the slope of the elliptic curve at P and draw the tangent line to E at P. So, let's figure out what the slope is. If our elliptic curve E is given by $y^2 = x^3 + ax + b$, then by implicit differentiation, the slope of the tangent line to E at some point (x_0, y_0) is $\frac{3x_0^2+a}{2y_0}$. This is not so good if $y_0 \equiv 0 \pmod{n}$, since we have to divide by y_0. And more generally, this is not so good when y_0 fails to be a *unit* modulo n, i.e., when $\gcd(y_0, n) > 1$. (In fact, it's not so bad when the denominator is 0, since then we end up with the identity, the point at ∞. But it's very bad if the denominator is nonzero but shares a factor with n.)

But "not so good" depends on your perspective: this is bad for *computing* $2P$, but we don't really want to compute $2P$: we *really* want to factor n. And as soon as we've found some number $y_0 < n$ with $\gcd(y_0, n) > 1$, then we've just found a factor of n. So in fact, we *want* to fail to compute mP. (This is probably a good metaphor for life.)

Here is Lenstra's elliptic curve factorization algorithm [Len87], in general:

(1) Pick an elliptic curve E over $\mathbb{Z}/n\mathbb{Z}$ and a point $P = P_1$ on E.
(2) Choose a smoothness bound B.
(3) For each prime q, let $c_q = \lfloor \log_q(n) \rfloor$.
(4) Let $m = 2^{c_2}3^{c_3}\cdots q^{c_q}$, where q is the largest prime $\leq B$.
(5) For each prime $q_i \leq B$, let $P_{i+1} = q_i^{c_{q_i}} P_i$, if this multiplication is possible. If at some stage the multiplication is impossible, because the denominator of some slope shares a factor with n, then we have found a factor of n and we're done! Otherwise, continue on with the next prime.

Example Let us factor $n = 45857027$, which is the product of two primes. Consider the elliptic curve $E : y^2 = x^3 + x + 6$. Consider the point $P = (4870940, 8616998)$ on E. Instead of taking a smoothness bound to be known in advance, let us just try to perform step (5) in the elliptic curve factorization algorithm until we fail.

We first find $c_2 = \lfloor \log_2(n) \rfloor = 25$. So we try to multiply P by 2^{25}. This is fine, and Sage happily tells us that $P_2 = 2^{25}P = (30411685, 10089499)$. So, no problems yet. Now, we compute $c_3 = \lfloor \log_3(n) \rfloor = 16$. So we compute $P_3 = 3^{16}P_2 = (34140465, 11984090)$. Again, no problems.

Next, we compute $c_5 = \lfloor \log_5(n) \rfloor = 10$. So we now compute $P_4 = 5^{10}P_3 = \cdots$. Uh oh: something didn't work, and Sage raises a scary error. Let us analyze what happened. We find that $5P_3$ is fine, and it is equal to $Q = (31863757, 4846021)$. But then we want to compute $5Q$. We do this by the successive doubling method: computing $2Q = Q + Q$, then $4Q = 2Q + 2Q$, then finally $5Q = 4Q + Q$. $2Q$ is fine: it is $(6004492, 6574745)$. Also, fine is $4Q = (5817393, 35259580)$. But as part of the process to compute $5Q$, we have to compute the slope of the line through $4Q$ and Q, which is $-\frac{30413559}{26046364}$. What element is this in $\mathbb{Z}/n\mathbb{Z}$? It's nothing because the denominator 26046364 is not an element of $(\mathbb{Z}/n\mathbb{Z})^{\times}$: it fails to be relatively prime to n. Indeed, $\gcd(26046364, n) = 587$. And we have learned that 587 is a factor of n. The other factor is 78121, which is also prime. So we have factored n.

Let us now suppose that $n = pq$. Then, this factorization method succeeds if the order of P modulo p in $E(\mathbb{F}_p)$ divides m, or if the order of P modulo q in $E(\mathbb{F}_q)$ divides m, but not both. Since the order of P modulo p in $E(\mathbb{F}_p)$ divides $\#E(\mathbb{F}_p)$, we really want $\#E(\mathbb{F}_p)$ to be B-smooth, but not $\#E(\mathbb{F}_q)$ (or vice versa). In the example above, $\#E(\mathbb{F}_{587}) = 600 = 2^3 \times 3 \times 5^2$, which is a 5-smooth number. Thus, it was *guaranteed* that the process would work if we choose any smoothness bound $B \geq 5$.

Of course, we don't know whether $\#E(\mathbb{F}_p)$ is going to be B-smooth. However, since smooth numbers are fairly common, we can hope to find a suitable elliptic curve just by trying a bunch at random.

Remark 15.2 In fact, we ought to know something about the *distribution* of $\#E(\mathbb{F}_p)$ as E varies over elliptic curves over $\mathbb{F}_p$. We know from the Hasse–Weil bound that $|\#E(\mathbb{F}_p) - p - 1| \leq 2\sqrt{p}$. But if this bound is not actually realistic, and (say) $\#E(\mathbb{F}_p)$ usually only took on very few values, then this would make the elliptic curve factorization algorithm not particularly useful: we probably wouldn't be able to find an E so that $\#E(\mathbb{F}_p)$ were smooth. Fortunately, this is not the case, and we understand the distribution of $\#E(\mathbb{F}_p)$ quite well: it follows a "semicircular" law, as follows: consider the quotient

$$c(E) = \frac{\#E(\mathbb{F}_p) - p - 1}{2\sqrt{p}}.$$

Then $-1 \leq c(E) \leq 1$, so let us write $c(E) = \cos(\theta_E)$ for some $0 \leq \theta_E \leq \pi$. Then, for any $-1 \leq \alpha < \beta \leq 1$ and p sufficiently large, we have

$$\frac{\#\{E : \alpha \leq \theta_E \leq \beta\}}{\#E/\mathbb{F}_p} \approx \frac{2}{\pi} \int_{\alpha}^{\beta} \sin^2 \phi \, d\phi.$$

See [Bir68]. The denominator on the left is the total number of elliptic curves over $\mathbb{F}_p$, which is slightly less than p^2 (since a few of the curves $y^2 = x^3 + ax + b$ aren't elliptic curves because $4a^3 + 27b^2 = 0$).

In practice, Lenstra's factoring algorithm is very good for numbers of moderate size, say around 40 digits.

15.4 Elliptic Curves Over $\mathbb{Q}$

Elliptic curves are important for far more than just cryptography. Elliptic curves over the rational numbers $\mathbb{Q}$ and the integers $\mathbb{Z}$ are ubiquitous in number theory today. (See problem 6 for an example of their use.)

The first interesting theorem about the structure of the rational points on an elliptic curve E is the Mordell–Weil Theorem.

Theorem 15.3 (Mordell–Weil) *Let E be an elliptic curve over $\mathbb{Q}$. Then, there are integers r and s with $r \geq 0$ and $0 \leq s \leq 2$, and points $P_1, \dots, P_r, Q_1, \dots, Q_s \in E(\mathbb{Q})$ and integers $d_1, \dots, d_s$ such that, if R is any point in $E(\mathbb{Q})$, then R can be expressed* uniquely *as*

$$m_1 P_1 + \cdots + m_r P_r + n_1 Q_1 + \cdots + n_s Q_s,$$

where the m_i's are integers, and $0 \leq n_j < d_j$.

The statement of this theorem is slightly unwieldy because we have not discussed all the group theory necessary to make it have a prettier formulation. If you have seen group theory, then we can express it as follows:

Theorem 15.4 (Mordell–Weil [Mor22]) *Let E be an elliptic curve over $\mathbb{Q}$. There is a nonnegative integer r and a finite abelian group T such that*

$$E(\mathbb{Q}) \cong \mathbb{Z}^r \times T.$$

We call r the *rank* and T the *torsion subgroup*.

Mazur extended the Mordell–Weil Theorem by providing the possible choices for the d_j's:

Theorem 15.5 (Mazur, [Maz77]) *If E is an elliptic curve over $\mathbb{Q}$, then there are 15 possibilities for the torsion subgroup of E. They are $\mathbb{Z}/n\mathbb{Z}$ for $1 \leq n \leq 12$ and $n \neq 11$, and $\mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/2n\mathbb{Z}$ for* $n = 1, 2, 3, 4$.

These are quite challenging theorems (especially Mazur's), and the proofs are beyond the scope of this book.

Mazur's theorem takes care of the torsion subgroup quite nicely: there are 15 possibilities, and all of them are torsion subgroups for infinitely many elliptic curves. Furthermore, there are algorithms available for computing the torsion subgroups. We will soon see the Nagell–Lutz Theorem, which gives us a finite set of possibilities for the torsion points, and then we can simply check all of them.

The rank, on the other hand, is much more mysterious. Noam Elkies has found elliptic curves of rank (at least) 28, but it is an open problem whether the rank can be arbitrarily high.

However, these high-rank elliptic curves are rare. Indeed, there is a conjecture that for 50% of elliptic curves over $\mathbb{Q}$, the rank is 0, and for 50% of them, the rank is 1. (There are, of course, curves with rank ≥ 2, and even infinitely many of them, but they occur 0% of the time. This is similar to the case of primes and composites: 100% of numbers are composite, but there are still plenty of primes in the remaining 0%.)

The mathematical community is split on the issue of whether ranks are expected to be bounded or not. Recently, Jennifer Park, Bjorn Poonen, John Voight, and Melanie Matchett Wood [PPVM16] came up with a heuristic that suggests that there should be an absolute bound on the rank of an elliptic curve, and that there should only be finitely many elliptic curves of rank greater than 21.

While we cannot *prove* all that much about the rank of an elliptic curve, there is a very famous and important open problem concerning this topic. It is one of the Clay Math Millennium problems, and it is known as the Birch and Swinnerton–Dyer Conjecture. This gives us a conjectural way of computing the rank. However, it would take a long time even to state the Birch and Swinnerton–Dyer Conjecture, so we will move on.

It sometimes happens that even the "smallest" rational points on elliptic curves (measured in terms of the size of the denominator of the x-coordinate, say) are very large.

Example Let E be the elliptic curve $y^2 = x^3 + 877x$. Then the rational point other than $(0, 0)$ with the smallest denominator on E has x-coordinate

$$\left(\frac{612776083187947368101}{7884153586063900210}\right)^2.$$

In the case of elliptic curves of rank 1, there is a formula due to Gross and Zagier [GZ86] for finding the nontrivial rational point of smallest denominator. See, for instance, [Wat12] for an algorithm.

15.5 Elliptic Curves and Diophantine Equations

A *Diophantine equation* is an equation that is intended to be solved in integers (or perhaps rational numbers). One of the most famous Diophantine equations is the Fermat equation. Let $n \geq 3$ be an integer. Then the nth Fermat equation is $x^n + y^n = z^n$. Fermat's Last Theorem says that there are no solutions to the Fermat equation in positive integers. This was finally proven by Wiles in 1995, more than 350 years after it was first stated. Elliptic curves played an essential role in Wiles's proof.

Let us see a simpler (but still very nontrivial) example of elliptic curves showing up in a Diophantine equation. We will study the following problem, known as the *congruent number problem*: given an integer n, determine whether there is a right triangle with rational sides and area n. Such a number is called a *congruent number*.

For example, if $n = 6$, then the answer is "yes": the right triangle with sides 3, 4, and 5 has area 6. Thus 6 is a congruent number.

If $n = 5$, then it is harder to come up with an example. However, some thought(!) produces the triangle with legs $\frac{20}{3}$ and $\frac{3}{2}$, which has area 5 and hypotenuse

$$\sqrt{\left(\frac{20}{3}\right)^2 + \left(\frac{3}{2}\right)^2} = \frac{41}{6}.$$

Of course, you undoubtedly noticed that $n = 157$ is a congruent number, since the triangle with legs

$$\frac{6803298487826435051217540}{411340519227716149383203} \text{ and } \frac{411340519227716149383203}{21666555693714761309610}$$

has hypotenuse

$$\frac{224403517704336969924557513090674863160948472041}{8912332268928859588025535178967163570016480830}$$

and area 157.

Oh, you didn't just notice that on your own? How can one find such crazy numbers? Using elliptic curves! We won't go through all the details of that computation here, but only some of the general points.

Let us *attempt* to solve the congruent number problem. We are looking for rational numbers a and b such that $\frac{1}{2}ab = n$ and $a^2 + b^2 = h^2$ for some rational number h. Replacing b with $\frac{2n}{a}$, we have the equation

$$a^2 + \frac{4n^2}{a^2} = h^2,$$

or

$$\frac{4n^2}{a^2} = h^2 - a^2 = (h - a)(h + a).$$

Next, let $g = h + a$. Then we have

$$h - a = \frac{4n^2}{a^2 g}.$$

Thus we have

$$2a = g - (h - a) = g - \frac{4n^2}{a^2 g}.$$

Multiplying by $\frac{1}{8}a^3 g^2$, we get

$$\frac{a^4 g^2}{4} = \frac{1}{8}a^3 g^3 - \frac{n^2 a g}{2}.$$

Letting $x = \frac{1}{2}ag$ and $y = \frac{1}{2}a^2g$, we get

$$y^2 = x^3 - n^2x,$$

which is an elliptic curve! Note that $a = \frac{y}{x}$.

Remark 15.6 This is the sort of way that elliptic curves frequently show up when working with Diophantine equations. Any equation of the form $y^2 = q(x)$, where $q(x)$ is a degree-4 polynomial with no repeated roots, can be massaged into the Weierstraß form of an elliptic curve, using similar substitutions. Similarly, if $f(x, y)$ is a cubic polynomial in x and y (meaning that for every monomial of f, the sum of the x-degree and the y-degree is at most 3), then f can be massaged into Weierstraß form. (However, it might not be an elliptic curve, for it might be the case that $4a^3 + 27b^2 = 0$.) At the beginning of the congruent number computation, we saw the expression

$$a^2 + \frac{4n^2}{a^2} = h^2$$

with variables a and h (n is fixed), so we could have *known* that there was an elliptic curve lurking, since we can easily rewrite it as

$$a^2h^2 = (ah)^2 = a^4 + 4n^2,$$

which is of the $y^2 = q(x)$ form. If we have an equation of the form $y^2 = f(x)$, where the degree of f is at least 5 and f has no repeated roots, then we *definitely* cannot turn it into an elliptic curve using reversible substitutions. Occasionally, we *can* do so using higher degree substitutions, and the question of when we can do so and how was a large part of my research for my PhD, although it was stated rather differently.

Now, that we have written down an elliptic curve, it's time to start understanding its solutions and translating them back into triangles. First, since $x^3 - n^2x = x(x - n)(x + n)$, we have some obvious solutions: $(0, 0)$, $(n, 0)$, and $(-n, 0)$. However, these solutions are not very helpful, since $a = y/x = 0$; they do not correspond to triangles. One can also check that there are no other torsion points on this elliptic curve. Thus, if n is to be a congruent number, then the elliptic curve $y^2 = x^3 - n^2x$ must have rank at least 1. Conversely, if the rank of the congruent number elliptic curve is at least 1, then n is a congruent number. But how can we tell? Well, there's an interesting theorem of Tunnell that makes checking relatively straightforward.

Theorem 15.7 (Tunnell, [Tun83]) *Suppose n is a squarefree positive integer, i.e., n is not divisible by the square of any prime. Define functions f, g, h, k as follows:*

$$\begin{aligned}
f(n) &= \#\{(x, y, z) \in \mathbb{Z}^3 : x^2 + 2y^2 + 8z^2 = n\},\\
g(n) &= \#\{(x, y, z) \in \mathbb{Z}^3 : x^2 + 2y^2 + 32z^2 = n\},\\
h(n) &= \#\{(x, y, z) \in \mathbb{Z}^3 : x^2 + 4y^2 + 8z^2 = n/2\},\\
k(n) &= \#\{(x, y, z) \in \mathbb{Z}^3 : x^2 + 4y^2 + 32z^2 = n/2\}.
\end{aligned}$$

If n is odd and n is congruent, then $f(n) = 2g(n)$. If n is even and n is congruent, then $h(n) = 2k(n)$. If the Birch and Swinnerton–Dyer Conjecture is true, then both of the converses hold.

Another related result is as follows:

Theorem 15.8 (Stephens, [Ste75]) *If the Birch and Swinnerton–Dyer Conjecture is true, then for any $n \equiv 5, 6, 7 \pmod 8$, the elliptic curve $y^2 = x^3 - n^2x$ has odd rank. This implies that the rank is positive, so that n is a congruent number.*

15.6 Elliptic Curves Over $\mathbb{Z}$

The *integer* points on an elliptic curve behave rather differently from the rational points. For one thing, $E(\mathbb{Z})$ is typically not a group. (See problem 1.)

Still, we can say something about the structure of $E(\mathbb{Z})$. The main theorem on the topic is due to Siegel:

Theorem 15.9 (Siegel, [Sie14]) *If E is an elliptic curve over $\mathbb{Z}$, then $E(\mathbb{Z})$ is a finite set.*

Siegel's original proof did not (and could not be adapted to) say anything about *how many* integral points an elliptic curve could have. Indeed, elliptic curves can have arbitrarily many integral points. (See problem 2.) But more recent results in transcendental number theory(!), in particular, Baker's Theorem on linear forms in logarithms, have made it possible to get a bound on the number of integral points an elliptic curve $y^2 = x^3 + ax + b$ has, in terms of a and b.

In fact, Siegel proved an even stronger theorem than Theorem 15.9. What he showed is that if P is a point on E, then the numerator and denominator of the x-coordinate of P have roughly the same number of digits. More precisely:

Theorem 15.10 (Siegel) *For a point $P \in E(\mathbb{Q})$, let us write its x-coordinate as $\frac{a(P)}{b(P)}$, in lowest terms. Then,*

$$\lim_{\substack{P\in E(\mathbb{Q}) \\ \max(|a(P)|,|b(P)|)\to\infty}} \frac{\log|a(P)|}{\log|b(P)|} = 1.$$

Theorem 15.10 implies Theorem 15.9 because if E were to have infinitely many integral points, then they would go off to ∞, so $a(P)$ (and hence $\log|a(P)|$) would go off to infinity, but $b(P)$ (and hence $\log|b(P)|$) would not.

Another very important theorem about elliptic curves tells us what the torsion points can look like.

Theorem 15.11 (Nagell–Lutz, [Lut37]) *Let E be the elliptic curve $y^2 = x^3 + ax + b$, with $a, b \in \mathbb{Z}$. Let $\Delta = 4a^3 + 27b^2$. If $P = (x, y)$ is a torsion point on E (i.e., $nP = \infty$ for some positive integer n), then x and y are both integers, and one of the following two things happens:*

(1) $2P = \infty$, *in which case either* $P = \infty$ *or* $y = 0$, *or*
(2) $2P \neq \infty$, *in which case* y *divides* Δ.

In fact, in the second case, y^2 divides Δ. The Nagell–Lutz Theorem thus gives us an algorithm for computing the torsion points on an elliptic curve over E. We relegate the proof of the Nagell–Lutz theorem to problems 17 and 18.

15.7 Problems

(1) Show that if E is an elliptic curve over a field K, then $E(K)$ is a group. Show that $E(\mathbb{Z})$ need not be a group by giving an example of an elliptic curve with two integral points whose sum is not integral.
(2) Show that for any integer n, there is an elliptic curve E over $\mathbb{Z}$ such that $\#E(\mathbb{Z}) \geq n$.
(3) Factor 377681131 using the Pollard $p - 1$ method.
(4) Factor 448866199 using the Lenstra elliptic curve method. (Hint: Hfr gur ryyvcgvp pheir l fdhnerq rdhnyf k phorq cyhf k cyhf bar.)
(5) Show that if $n \equiv 5, 6, 7 \pmod 8$ is squarefree, then n satisfies Tunnell's criterion of Theorem 15.7.
(6) A set $\{a_1, a_2, \ldots, a_m\}$ of integers is said to be a Diophantine m-tuple if for any distinct $i, j, a_i a_j + 1$ is a perfect square. Show that if $\{a, b, c\}$ is a Diophantine triple, then there are only finitely many integers d such that $\{a, b, c, d\}$ is a Diophantine quadruple. (There are infinitely many Diophantine quadruples, such as $\{1, 3, 8, 120\}$. It was recently shown in [HTZ16] that there are no Diophantine quintuples).
(7) Let $n > 1$, and suppose E is an elliptic curve over $\mathbb{Z}/n\mathbb{Z}$. Suppose that there are distinct primes $\ell_1, \ldots \ell_k$ and points $P_i \in E(\mathbb{Z}/n\mathbb{Z})$, not equal to the identity, such that

    (a) $\ell_i P_i = \infty$ for all i, and
    (b) $\prod_{i=1}^{k} \ell_i > (\sqrt[4]{n} + 1)^2$.

    Show that n is prime. (Hint: Fubj gung sbe rirel cevzr c qvivqvat a, rnpu y qvivqrf gur ahzore bs cbvagf ba R bire c.) This is a primality test based on elliptic curves, an elliptic curve version of the Pocklington–Lehmer test.
(8) Suppose that E is the elliptic curve $y^2 = x^3 + ax$, and suppose that p is a prime congruent to $3 \pmod 4$. Show that $\#E(\mathbb{F}_p) = p + 1$.
(9) Let E be the elliptic curve $y^2 = x^3 + ax + b$ over $\mathbb{F}_p$, with $p \geq 5$. Suppose that $d \in \mathbb{F}_p^\times$ is not a perfect square, i.e., there is no $r \in \mathbb{F}_p$ such that $d \equiv r^2 \pmod p$.

    (a) Convert $dy^2 = x^3 + ax + b$ to short Weierstraß form. Call this curve E'.
    (b) Express $\#E'(\mathbb{F}_p)$ in terms of $\#E(\mathbb{F}_p)$.
    (Hint: Rknpgyl unys gur abamreb ryrzragf bs gur svryq jvgu c ryrzragf ner cresrpg fdhnerf).

(10) Let E be the elliptic curve $y^2 = x^3 + ax + b$ over some field F. Recall that if $P = (x, y)$ is a point on E, then the x-coordinate of $2P$ is

$$f(x) = \frac{x^4 - 2ax^2 - 8bx + a^2}{4(x^3 + ax + b)}.$$

Show that (x, y) is a torsion point of $E(F)$ if and only if the sequence

$$x, f(x), f(f(x)), f(f(f(x))), \ldots$$

eventually repeats. This observation is one of the motivations behind the subject of *arithmetic dynamics*. The standard algorithm in arithmetic dynamics is to turn theorems about elliptic curves into impossibly difficult general conjectures about such dynamical systems.

(11) The right triangle with sides 3, 4, and 5 has area 6. Using elliptic curves, find three more right triangles with rational sides and area 6.

(12) Find a change of variables like the one used in Section 15.5 to transform the curve $x^3 + y^3 = d$ into an elliptic curve in short Weierstraß form.

(13) Let E be an elliptic curve with rational coefficients, and let P be a rational point of infinite order on E.

(a) Show that we can write $P = \left(\frac{a}{c^2}, \frac{b}{c^3}\right)$, where $\gcd(a, c) = \gcd(b, c) = 1$.

(b) Let c_n^2 be the denominator of the x-coordinate of nP. Show that (c_n) is a *strong divisibility sequence*, i.e., that $\gcd(c_m, c_n) = c_{\gcd(m,n)}$.

(c) Show that the Fibonacci numbers $F_0 = 0$, $F_1 = 1$, $F_n = F_{n-1} + F_{n-2}$ for $n \geq 2$ also form a strong divisibility sequence.

(14) (a) Show that there are infinitely many solutions to the equation $x^2 - 3y^2 = 1$, where x and y are integers. (This is an example of a *Pell equation*. More generally, a Pell equation is one of the form $x^2 - dy^2 = k$. If $k = 1$, then there are infinitely many solutions if $d > 0$ is *not* a perfect square.)

(b) Show that there are only finitely many solutions to the simultaneous Pell equations $x^2 - 3y^2 = 1$ and $x^2 - 7z^2 = 1$.

(15) (a) Let E be an elliptic curve over a field F. Show that $f : E \to E$ given by $f(x, y) = (x, -y)$ is an automorphism of E. (If G is a group, then a function $f : G \to G$ is said to be an *automorphism* if f is a bijection, and $f(gh) = f(g)f(h)$ for all $g, h \in G$.)

(b) Suppose E is an elliptic curve of the form $y^2 = x^3 + a$, over a field F. If $\zeta \in F$ is such that $\zeta^3 = 1$, show that $f(x, y) = (\zeta x, -y)$ is an automorphism of E.

(c) Suppose E is an elliptic curve of the form $y^2 = x^3 + ax$, over a field F. If $i \in F$ is such that $i^2 = -1$, show that $f(x, y) = (-x, iy)$ is an automorphism of E.

(16) If F is a finite field containing $\mathbb{F}_p$, where p is prime, and E is an elliptic curve over $\mathbb{F}_p$, show that $\sigma : E(F) \to E(F)$ given by $\sigma(x, y) = (x^p, y^p)$ is an automorphism of $E(F)$. (If $F = \mathbb{F}_p$, then σ is the identity, by Fermat's Little

Theorem.) σ is called the *Frobenius automorphism* and is extremely important in arithmetic geometry.

(17) Let E be an elliptic curve over $\mathbb{Z}$. Show with bare hands (i.e., not using the Nagell–Lutz Theorem) that if P is a rational point but not an integral point, then P has infinite order.

(18) Finish the proof of the Nagell–Lutz Theorem.

(19) Modify the Pollard $p-1$ factorization algorithm to work when $p+1$ is smooth. (This is hard.)

Chapter 16
Zero-Knowledge Proofs

16.1 Introduction to Zero-Knowledge Proofs

Generally in mathematics, the goal of a proof is for the writer to convey knowledge and understanding to the reader. (However, you may doubt this claim based on some books or papers you have read or talks you have attended, although surely not any of mine.) But from time to time, the prover *wants* to obfuscate some key pieces of information. As we shall see, being able to obfuscate some key pieces of information in the proof has important cryptographic implications.

Let us start with a toy example, which is taken from the charming paper [NNR99]. Many of us, when we were very young, did "Where's Waldo" puzzles. The goal is to find Waldo in a picture, which may contain similarly dressed decoys. Suppose you have found Waldo, and you want to convince me that you know where he is. You *could* just point at him, but where's the fun in that? Now, you have just ruined a perfectly good puzzle for me.

Instead, what you want to do, somehow, is to convince me, beyond a reasonable doubt, that you know where he is, while still not giving me any information about where to find him. How can you do this?

One possibility is that you could just *declare* that you have found him. But this is not a convincing evidence: you could just as easily claim that you know where Waldo is when you are, in fact, just as clueless as I.

In fact, there is a way that you can convince me that you have found Waldo, without giving away any information about where he is.

One possibility is as follows: you take the *Where's Waldo* book, make a photocopy of the page, cut out the Waldo in the photocopy (as civilized people, we do not cut up actual books), and show me the Waldo cut-out.

But I fundamentally don't trust you. I suspect that you have photocopied a *different* page of the *Where's Waldo* book, one which is much easier, and cut out the Waldo from that page. You could do that with a bit of sleight of hand, and I would be none the wiser.

S. Rubinstein-Salzedo, *Cryptography,* Springer Undergraduate Mathematics Series, https://doi.org/10.1007/978-3-319-94818-8_16

So, I don't accept that solution. But there is another solution that would satisfy even me, at my most stubborn. What you can do is to take a giant piece of paper, much larger than the *Where's Waldo* book, cut out a small Waldo-sized hole in it, and put the hole over the Waldo. Then I can verify that you have actually found Waldo, since I can easily verify that the character in question is actually Waldo.

Finally, one more step you should take: pull the book out from under the paper (while covering the hole), so that I can be satisfied that you haven't flipped the page while navigating Waldo to the hole.

An interesting consequence of all this is that, although I am now entirely convinced that *you* know where Waldo is, and indeed that the page does, in fact, contain a Waldo, I will struggle to convince anyone else of this fact. I could videotape the entire process of you putting the paper over the Waldo book and showing me the Waldo, point the camera at the Waldo once it has been covered by the hole, and so forth. But a skeptical third party would not be satisfied: I could have paused the video at several points in the middle and flipped to a different page in the *Where's Waldo* book, and someone replaying the video would have no method of detecting the trick. Video editing software is powerful!

16.2 What is a Zero-Knowledge Proof?

Let us now abstract this idea a bit and talk about what a zero-knowledge proof is supposed to do, in general. As in standard cryptography, we have some characters, but they aren't the same characters we met earlier. In zero-knowledge proofs, we typically have two characters, named Peggy (the prover) and Victor (the verifier). Peggy knows some piece of information, and she wants to convince Victor that she knows that particular piece of information, while giving away absolutely nothing else. Typically, this piece of information is hard to come up with in the first place. (There isn't much point in having Peggy convince Victor that she knows the answer to a problem, when he can solve the problem easily himself. We're all mature people here; we don't have to tease and examine each other.) As we have already discussed quite a lot, factoring appears to be a difficult problem. If Peggy knows the prime factorization of some large number n, can she somehow convince Victor that she knows it, without giving away any details of the factorization? (For example, she should not be willing to give away any information about the *number* of prime factors, or how many digits the smallest prime factor is, or any other related information.)

Generally, a zero-knowledge proof will consist of several rounds of information exchange. For example, Peggy may do some secret precomputation and give the result of the precomputation to Victor. Victor can then use the result of this precomputation to issue a challenge to Peggy, a challenge that Victor knows how to solve. (Or, if he can't solve it on his own, he has some way of verifying after the fact that Peggy did it correctly.) Using her secret information, Peggy solves Victor's challenge problem and reports the answer. This exchange can go on for several rounds, until Victor is convinced that Peggy does indeed possess the secret information.

Now, Victor may be skeptical, but he is also a reasonable and decent human being. (On the other hand, we do *not* assume that Peggy is a reasonable and decent human being. It is not a good idea to trust people who can do magic calculations and come up with answers to computationally difficult problems.) Victor does not demand absolute certainty that Peggy has the secret information; for example, she might just be able to guess the answers to all of Victor's challenges completely by accident, without needing to know the secret information. Or, she might attempt to cheat by guessing Victor's challenges and solving easier problems that will allow her to give seemingly correct answers to the challenges. (We'll see an example of this later.) But, assuming that Victor chooses suitable challenges, Peggy's chances of guessing correctly every time are very small, small enough that Victor realizes that it is *much* more likely that she is being honest, than that she is lying and getting lucky every time. If Peggy claims that she can flip a coin 1000 times in a row and get heads every time, *and then she does so*, it's more likely that she has a two-headed coin than it is that that happened just by chance, isn't it?

We expect a zero-knowledge proof to satisfy three important properties:

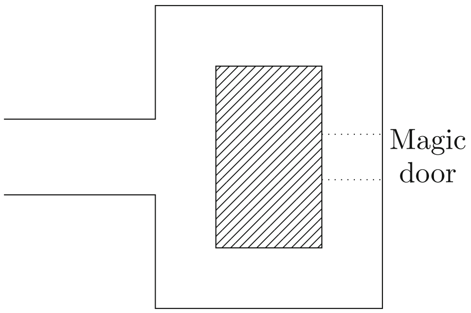
Completeness: If Peggy is telling the truth, then Victor will be convinced of this.
Soundness: If Peggy is lying, then she cannot convince Victor that she is telling the truth, except with some very small probability.
Zero-knowledge: If Peggy is telling the truth, then Victor learns nothing other than the fact that she is telling the truth.

Remark 16.1 Formally, the zero-knowledge property is quite subtle. We have the notion of an *honest verifier*, who follows the proof protocol with the prover, as well as a *cheating verifier*, who creates a transcript that could potentially be that of the communication between an honest verifier and the prover. The zero-knowledge property says that a real transcript, created by an honest verifier and the prover, is indistinguishable from a transcript that is entirely fabricated by a cheating verifier. The reason this is a desirable property is that we do not wish to allow the verifier to pretend to be the prover at some later point.

Remark 16.2 In fact, even Remark 16.1 is *still* not fully rigorous, because we have not stated what we mean by two transcripts being "indistinguishable." One possibility is that the fake transcript could actually *be* a real transcript. But a weaker notion that is sometimes useful is a computational version of indistinguishability: it might not be possible for the fake transcript to occur as a result of a conversation between an honest verifier and an honest prover, but it takes a third party a very long time to detect the forgery. This is usually also acceptable to us.

Let us see more examples of zero-knowledge proofs.

Figure 16.1 Ali Baba's Cave



16.3 Ali Baba's Cave

In addition to playing Where's Waldo as kids, we also learn about Ali Baba. He followed a band of thieves to learn of a treasure-filled cave with an entrance that would open upon hearing the magic words "Open Sesame!"

In the zero-knowledge story of Ali Baba's Cave, as told by Quisquater and Guillou in their also-charming paper [QGAB89], there is a cave with two entrances. (See Figure 16.1.) There is a door connecting the two entrances, and this door will open to a person who says the words "Open Sesame!" and otherwise it remains shut.

Ali Baba brings a friend, Vera, to the cave and shows her the two entrances. He tells her that there are certain magic words he can say to open a door between the entrances. Naturally, Vera is skeptical. (Wouldn't you be, especially if you were living in the Middle Ages and there was no such thing as voice recognition software?) So, she asks Ali Baba to indulge her in a verification. Ali Baba is to enter the cave through one of the two entrances, of his choosing. Then Vera flips a coin, and depending on the result of the coin flip, she either calls out "Come out through the left entrance" or "Come out through the right entrance." Since Ali Baba can get through the door, he can come out through the desired entrance, regardless of which side he entered.

Now, one attempt doesn't prove much. So, they repeat this experiment 40 times. Every time, Ali Baba comes out through the correct entrance. If he didn't have access to the door, then the chances of his being able to do this would be $\frac{1}{2^{40}}$, which is very small. Vera believes that Ali Baba does indeed know the magic words.

Maybe you object to this procedure: wouldn't it be easier if Vera were to *watch* Ali Baba enter the cave through the left side and then come out through the right side? That would eliminate the need for repeating the test over and over again: if Ali Baba can enter through the left side and exit through the right side even once, then he *must* know the magic words. (We're ignoring the possibility of quantum tunneling here.)

Yes, this would be easier, but the other version is better. Why? If they adopted this second procedure, and Vera were to videotape the process, then she could show her videotape to a third party. (Suppose they haven't invented video editing software or anything similar yet.) Vera could then convince this third party that Ali Baba knows the magic words. But we don't want to allow her to do that. In terms of honest and cheating verifiers, this procedure would allow her to create a "transcript" of the event that she could not make on her own, without Ali Baba's help.

In the first version, a videotape of the experiment is not a convincing evidence that Ali Baba knows anything. The entire process could have been staged, so that Ali Baba and Vera agreed in advance which side Ali Baba should exit (and hence enter).

But what about the coin flips? Aren't those enough to make it random and hence convincing? First of all, with a bit of practice, one can flip a coin in such a way that one knows what the outcome will be in advance. (One way to do this is to "flip" the coin so that it only wobbles and never actually flips over. It is not so easy for a spectator to tell the difference. See [DHM07].)

But more seriously, in cryptography, "random" numbers are not really so random: they are the result of a deterministic process that starts with a given "seed" value. So, it is possible that Ali Baba and Vera have set their seed value in advance, so that they know what all the "coin flips" are going to be. There *are* sources of true randomness such as radioactive decay, but your computer most likely isn't using them. Sometimes this can be a serious problem; for example, you can read about how some people managed to cheat an online poker server through bad random number generation, among other serious errors [AHM+99].

16.4 Three-Colorability

Let us now look at some less frivolous and more potentially relevant zero-knowledge proofs. In mathematics, the term "graph" has several different meanings, and they are unrelated to each other. For us, a graph will consist of a set V of vertices, and a set E of edges. An edge consists of an unordered pair of distinct vertices. Frequently, we represent a graph as a picture, with vertices pictured as dots and edges pictured as lines or curves connecting their two vertices. (See Figure 16.2.)

Graph theory is a large subject, and there are many things that are known about graphs. One of the most famous results about graphs is the infamous four-color

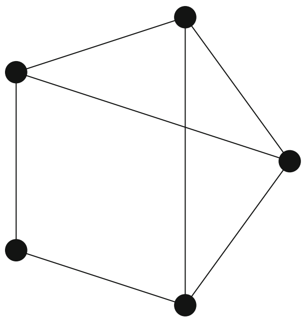


Figure 16.2 A graph

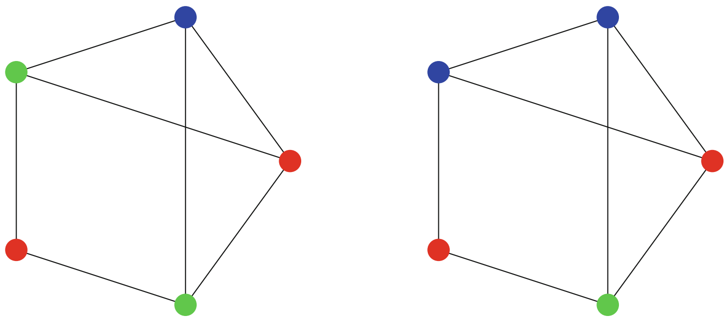
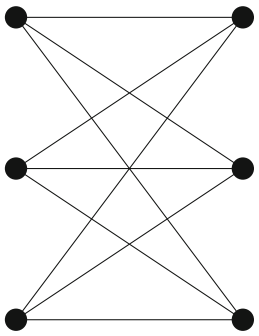


Figure 16.3 Left: A valid coloring with three colors. Right: An invalid coloring, since there are two adjacent blue vertices

Figure 16.4 This graph, known as $K_{3,3}$, is not a planar graph



theorem. We are interested in coloring each vertex of the graph with one of several colors, in such a way that if two vertices are connected by an edge, then they must be colored using different colors. See Figure 16.3 for a valid coloring with three colors, as well as an invalid coloring. A graph is said to be k-colorable if it can be colored in this way using (at most) k colors. A graph is said to be *planar* if it is possible to draw it on the plane without any edge crossings. A famous example of a nonplanar graph is the graph known as $K_{3,3}$, pictured in Figure 16.4.

Theorem 16.3 (Four-Color Theorem, [AH77, AHK77]) *Every planar graph is 4-colorable.*

The four-color theorem is notoriously difficult, and it was only proven in 1976 by Kenneth Appel and Wolfgang Haken after an extensive computer search, perhaps the first major theorem whose proof relied on computers in an essential way.

Now, *some* planar graphs can be colored using only three colors, but not all of them. Furthermore, given a graph G, it does not appear to be easy to determine whether G is 3-colorable. (In fact, this problem is NP-complete.)

In general, hard problems like this are good candidates for zero-knowledge proofs. In fact, any NP problem has a zero-knowledge proof. (Once we exhibit a zero-knowledge protocol for 3-colorability, the proof that 3-colorability is NP-complete will imply that any NP problem has a zero-knowledge proof. However, it also appears that problems that are not in NP can have zero-knowledge proofs; see problem 9 for an example of a zero-knowledge proof for a problem that does not appear to be in NP.) Let us see how it works in this case.

We assume that a graph *G is* three-colorable, and that Peggy knows a three-coloring. She would like to convince Victor that she knows a three-coloring, without giving away any information about how the coloring works. In particular, she does not want to tell him *how* to do the three-coloring.

Why might she not want to tell? Well, perhaps she eventually wants to sell him the three-coloring. She doesn't want to tell him how it works before he pays, because once he knows the three-coloring, he has no incentive to pay for it. On the other hand, if Victor pays up front, then he will be unhappy if Peggy is lying and doesn't actually know how to three-color the graph, or if the graph turns out not to be three-colorable at all. From a legal perspective, it is possible to set up an escrow system to deal with such practical issues, but isn't it just more fun to solve all our problems using math?

So, now for our procedure. For each edge in the graph, Victor needs to know that the two vertices it connects are colored differently. So, he can pick an edge and ask Peggy for the colors of its two vertices. If the two colors are ever the same, then he knows that Peggy is lying.

However, there are several reasons why this is not yet a satisfactory solution. From Victor's perspective, it is not satisfactory, because Peggy can just say whatever she wants, regardless of the truth. From Peggy's perspective, it is also not satisfactory, because Victor is getting information about the coloring.

We can improve the situation from Victor's perspective by having Peggy first color the graph and then cover all the vertices. Then Victor gets to point to an edge, and Peggy uncovers the two vertices that it connects, so that Victor can see that the two vertices are colored differently.

But this is still not satisfactory from Peggy's point of view, since she is giving away information about the coloring. However, note that if she has a coloring of the graph, say with red, green, and blue vertices, then she can easily generate several more colorings. For example, she gets a new coloring by swapping all the red and blue vertices, and keeping all the green ones fixed.

In fact, starting from one 3-coloring, Peggy can very easily produce five more, for a total of six. We can use this to modify the earlier attempt: instead of always using the same coloring each time Victor asks for an edge, she picks a random one of the six colorings before each request.

More precisely: Peggy takes her six 3-colorings of the graph. They then repeat the following procedure several times: Peggy picks one of her six 3-colorings at random and covers all the vertices. Then Victor points to an edge. Peggy removes the covers from the two vertices of that edge. Victor checks that the two vertices are colored differently. After enough queries, Victor is confident that Peggy does in fact have a 3-coloring. He has learned nothing about the coloring, because if he sees that two vertices are colored (say) red and blue, then all that means is that there is a 3-coloring in which they *could* be colored red and blue. But this is obvious, because the specific colors have no special meaning.

If Peggy is cheating, meaning that she doesn't actually have a 3-coloring, then Victor will eventually figure this out, because he will point to an edge and see two adjacent vertices with the same color. Or perhaps Peggy has cheated by not coloring all the vertices. Victor will also eventually notice this.

Suppose that the graph has m edges. Then, if Peggy is lying, then there must be one "faulty" edge, with two vertices of the same color. So, if Peggy is lying and Victor picks a random edge, then his chance of exposing the lie is at least $\frac{1}{m}$. If they repeat the experiment m times, then his chance of ever exposing the lie is at least $1 - \left(1 - \frac{1}{m}\right)^m \approx 1 - \frac{1}{e}$. If they instead repeat it mk times, for some k, then his chance of finding out is at least $1 - \left(1 - \frac{1}{m}\right)^{mk} \approx 1 - \frac{1}{e^k}$, which becomes very close to 1 for large k. It is reasonable for them to be willing to do mk trials, since this is a polynomial function of m, the number of edges in the graph.

16.5 Sudoku

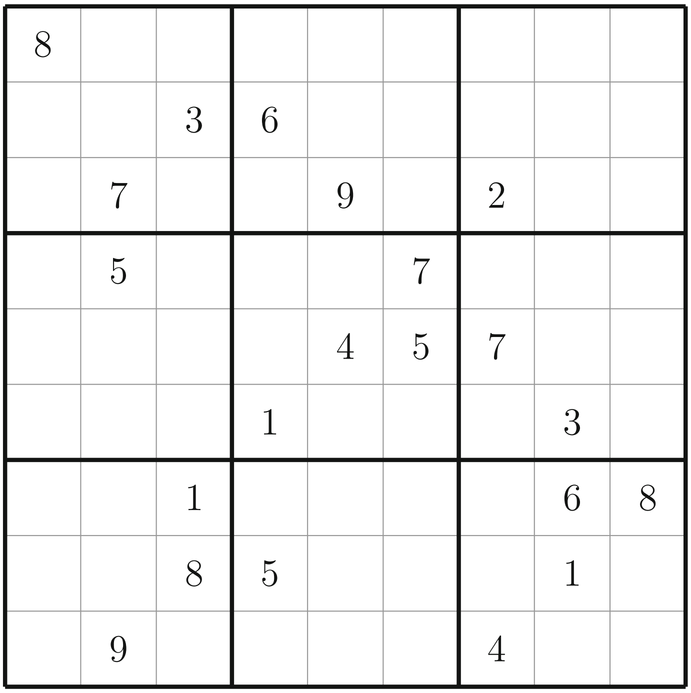
This time, Victor is trying to do a Sudoku and is having trouble. As usual, Peggy knows how to do it. Instead of giving a hint, she wants to rub it in and convince Victor that she already knows the solution.

In case you have not seen it before, here are the rules of Sudoku. There is a 9×9 grid of squares, divided into a 3×3 grid of 3×3 squares. Initially, some of the squares are filled in with numbers from 1 to 9. The goal is to fill in the remaining squares, in such a way that each row, column, and 3×3 square contains each of the digits from 1 to 9, exactly once. See Figure 16.5 for a challenging example [Col12].

How does Peggy demonstrate to Victor that she knows how to solve the Sudoku? Since Sudoku (at least, in its $n^2 \times n^2$ generalization) is an NP problem (in fact, like 3-colorability, it is NP-complete), we can convert the Sudoku problem to a 3-colorability problem and use the zero-knowledge proof we just looked at for 3-colorability. But such a proof would be difficult to understand.

Instead, we want a zero-knowledge proof that has more of a Sudoku "flavor" to it. Here is one. (Find another one in problem 15.) First, Peggy fills out her solution on a separate sheet, that she does not show to Victor. Now that she has this one filled-out grid, it is easy for her to generate others. Let σ be any permutation of the numbers $\{1, \ldots, 9\}$, i.e., a bijective function $\sigma : \{1, \ldots, 9\} \to \{1, \ldots, 9\}$. Recall that this means that for each $y \in \{1, \ldots, 9\}$, there is a *unique* $x \in \{1, \ldots, 9\}$ such that $\sigma(x) = y$. To generate a new completed Sudoku grid, she simply replaces every x on the grid with $\sigma(x)$. Since there are $9! = 362880$ permutations of $\{1, \ldots, 9\}$, this gives her 9! filled out Sudoku grids.

This suggests an attempted zero-knowledge proof. Peggy picks one of the 9! permutations at random and generates the corresponding filled-out grid. Victor needs



8								
		3	6					
	7			9		2		
	5				7			
				4	5	7		
			1				3	
		1					6	8
		8	5				1	
	9					4		

Figure 16.5 A difficult Sudoku problem

to verify that this is a correct solution, so he needs to check that, in every row, every number from 1 to 9 is used exactly once. Similarly for every column and 3×3 square.

So, we allow Victor to name a row, column, or 3×3 square. Peggy then reveals all the entries in that row, column, or 3×3 square, and Victor can check that all the numbers are indeed different.

While this is enough to convince Victor that Peggy has a complete Sudoku grid, it is not completely satisfactory. The reason is that Peggy could have chosen an entirely different (and much easier) Sudoku puzzle to start with. Victor also needs to know that Peggy started with the same numbers filled in as he did.

So, we need to allow Victor one more option. In addition to being allowed to see all the entries in a row, column, or 3×3 square, he also gets to ask to see what's in all the squares that were originally filled in. He doesn't expect to see exactly the same numbers as in the original, but he expects to see the same *pattern* as in the original: if he knows of two squares that (say) both contain 4's, then he expects to see the same number in both squares, even though they might not be 4's. Similarly, if he knows of two squares that contain *different* numbers, then in the version Peggy shows, they

1								
		4	8					
	9			2		6		
	5				9			
				7	5	9		
			3				4	
		3					8	1
		1	5				3	
	2					7		

Figure 16.6 A permutation of the original Sudoku problem

should also show different numbers. A permutation of the original Sudoku is shown in Figure 16.6.

After running this test several times, Victor is convinced that Peggy is telling the truth.

16.6 Quadratic Residuosity

As we have already seen, the problem of computing square roots modulo a composite number n is of cryptographic importance; for example, this was the idea behind the Miller–Rabin primality test. Furthermore, the problem of finding a square root modulo n is as hard as factoring n. (See problem 8.) Furthermore, if n is composite, then it is not even easy to tell if some number a is a square modulo n; that is, it is not easy to determine whether there exists some r so that $r^2 \equiv a \pmod{n}$. Note that this problem does not require actually finding r.

On the other hand, if p is prime, then it turns out that it is easy to determine whether a is a square modulo p, thanks to Euler's criterion:

Theorem 16.4 (Euler's criterion) *Let p be an odd prime and a not a multiple of p. Then*

$$a^{\frac{p-1}{2}} \equiv \begin{cases} +1 \pmod{p} & \text{if } a \text{ is a square modulo } p, \\ -1 \pmod{p} & \text{if } a \text{ is a nonsquare modulo } p. \end{cases}$$

We leave the proof for problem 3.

Observe that if $n = pq$ is a product of two primes, then a is a perfect square modulo n if and only if it is a perfect square modulo both p and q. (See problem 2.)

The way that Peggy and Victor can arrange a zero-knowledge proof for determining whether a is a square modulo n is as follows: Peggy will give Victor a choice of two closely related numbers, and Victor will ask Peggy to take the square root of one of them. If Victor could get information about the square roots of *both* numbers, then he could determine a square root of a, but he will never be given the opportunity.

Here is the protocol: Peggy knows a square root t of a modulo n. She picks a random number $b \in (\mathbb{Z}/n\mathbb{Z})^\times$ and computes $y = b^2 \pmod{n}$. She announces the value of y to Victor. Then Victor chooses whether to ask her to compute a square root of y, or of ay, modulo n. She responds to the query with some number c, which is either b or bt depending on the query, and he checks whether $c^2 \equiv y$ or $c^2 \equiv ay$, depending on his query. They repeat this procedure for several rounds, until Victor is convinced that Peggy is honest.

Now, after each value of y that Peggy announces, she must be prepared to provide square roots of both y and ay modulo n. But if she knows u and v such that $u^2 \equiv y \pmod{n}$ and $v^2 \equiv ay \pmod{n}$, then $u^{-1}v \pmod{n}$ is such that $(u^{-1}v)^2 \equiv a \pmod{n}$, i.e., Peggy also knows a square root of a.

Note that Peggy ought to be careful never to provide the same value of b (or y) twice, or else she gives Victor enough information to find t: upon hearing the same value of y twice, he once asks for a square root of y, and once for a square root of ay. He then performs the same computation as Peggy could have done in the preceding paragraph. If n is large and she picks b *randomly*, then it is unlikely that this problem will ever arise.

Since the problem of finding a square root modulo n is equivalent to factoring (see problem 8), we can treat this as a zero-knowledge protocol for factoring as well.

16.7 Discrete Logarithms and the Schnorr Protocol

Since 3-colorability is an NP-complete problem and we have a zero-knowledge proof for it, we know that there must be a zero-knowledge proof for every NP problem, including the discrete logarithm problem. However, the zero-knowledge proof we get after chasing around that rabbit hole probably won't be very enlightening. Instead, let's try something different.

We assume that we have a prime p and a primitive root g modulo p, and given $A \pmod p$, we wish to find a such that $g^a \equiv A \pmod p$. This is the standard discrete logarithm problem in $\mathbb{F}_p^\times$.

We suppose that Peggy knows the answer to this discrete logarithm problem, and she wishes to convince Victor of this fact. They employ the following procedure, known as the *Schnorr protocol.*

First, Peggy picks some number r and computes $t = g^r \pmod p$. Victor then challenges her by sending her a number c. Peggy responds with the number $s = r + ac$. Victor then checks that $g^s \equiv tA^c \pmod p$. If Peggy is honest, then this will be the case, for

$$g^s \equiv g^{r+ac} \equiv t \times (g^a)^c \equiv t \times A^c \pmod p.$$

Furthermore, it is easy for Victor to detect cheating, since any wrong value of s (at least, modulo $p-1$) will lead to the wrong answer. So, if Peggy instead chooses a different number b in place of a, she will only trick Victor if $r + ac \equiv r + bc \pmod{p-1}$, i.e., if $(a-b)c \equiv 0 \pmod{p-1}$. And remember that Victor chooses c; he can make sure that $\gcd(c, p-1) = 1$ if he chooses to do so.

Note that if they repeat the procedure, then Peggy needs to pick a new r each time. If she uses the same r twice, and Victor chooses $c = 1$ the first time and $c = 2$ the second time (say), then he knows the values of $r + a$ and $r + 2a$, and so he can compute a.

But there are tricks! Peggy may have access to magical information, so it is not too unreasonable to give her even more powers. Suppose that she can *guess* Victor's choice of c. Then she can cheat. She starts the same way, by choosing r. But now, instead of computing $t = g^r \pmod p$, she cheats and computes $t' = g^r/A^c \pmod p$. Victor, not suspecting anything, sends c, as predicted. Then she returns r, instead of s.

Now, what does Victor do? Still oblivious to the ruse, he computes g^r and checks that this is equal to $t' \times A^c$. This is true, since

$$t' \times A^c \equiv g^r/A^c \times A^c \equiv g^r \pmod p.$$

As a result, this procedure is only convincing for Victor if he is already convinced that Peggy isn't too powerful. From his point of view, it is good not to make his random choices until the last possible moment: if Peggy can predict his (potentially random) c, then she can cheat.

Can Victor gain information from this procedure? We don't know. We can prove that, assuming that the discrete logarithm problem is hard, he is unlikely to gain any knowledge by choosing his values of c at random. But what if he doesn't choose his values at random but instead adopts a strategy? Then we don't know: no one has suggested either a strategy whereby he can compute a, or even anything nontrivial about a such as $a \pmod 2$. But no one has provided a proof that he can't, either.

16.8 Time Machines and Prover Tricks

One day, you receive an unsolicited letter in the mail, predicting the winners of five sports games to take place over the next month. Instead of putting it in the recycling bin, where it belongs, you keep the letter and are surprised to find out that all five predictions are correct. The following month, you get another letter from the same person, again predicting the results of five sports games to take place over the next month. Now you are curious, so again you keep the letter. Again, you find that all five are correct. The following month, you get yet another letter from the same person, asking if you would like to pay for more predictions. Since the predictor has a perfect 10/10 record, maybe this is a good idea?

Not so fast! What you should try to determine is whether other people have received very similar letters. Here is a possible scenario: for the first mailer, the sender prepared 32 copies of each of 32 different letters, containing all possible results of the five games. Then the sender selected 1024 people and mailed a letter to each of those 1024 people. So, 32 people, including you, got the letter with all correct predictions. For the second letter, the sender wrote just 32 letters, each one containing a prediction for all five games the following month. The sender then mailed one of these letters to each of the people who had received the perfect mailer the previous month. The third letter, offering to sell predictions, was only sent to you, since you were the only one who got the perfect predictions the first two times. (Maybe the people who got letters with 9/10 right also got this third letter. After all, 9/10 is pretty good too!)

In other words, the sender has no ability whatsoever to predict. But you don't know that, because you only got to see one letter, out of all 1024. Therefore, you should be extremely suspicious of such advertising if you do not know the mechanism by which it was produced.

Or, take the following similar scenario. Let us suppose, for some rather unfathomable reason, I feel the need to make a lot of money in a short amount of time. My training as a mathematician has not taught me much about how to do this, so I do the logical thing under the circumstances: I plan to ask the experts. I assemble a list of many people who have made a lot of money in a short amount of time, and I arrange to interview all of them about their behavior and lifestyle choices. Then, I interview them all and look for patterns. And I find a very noticeable pattern in their behaviors shortly before they made their fortunes. Naturally, I choose to mimic their behavior: I buy a lottery ticket.

Oops. That wasn't what we wanted. My mistake was that I failed to interview all the people who did *exactly the same thing* but didn't win. I was the victim of survival bias.

In the context of zero-knowledge proofs, a dishonest Peggy may be able to pull a similar stunt. She doesn't know the secret she's supposed to know, but she wants to convince someone that she does anyway. So, she plays one of these probabilistic games, in which she can only trick Victor with small probability when she doesn't know the secret. Since she doesn't know it, she probably fails. But she is undeterred:

she just tries again, this time with a different verifier. She probably fails again, so then she moves on to a third verifier. Eventually, just by chance, she manages to trick someone. Success!

Similarly, a Peggy with a time machine can pull off the stunt with against a single verifier. Whenever Victor rejects her claim of knowing the secret, she uses her time machine to "undo" the last interaction. We conclude that Victor's confidence in a zero-knowledge proof should depend on what he believes Peggy's power to be. If he thinks she has a time machine, then he won't accept the protocols we have already discussed, and they have to come up with a more convincing protocol. We leave it as an exercise to the reader to try to come up with time-machine-resistant zero-knowledge proof. (See problem 18.)

16.9 Commitment Schemes

Alice and Bob are having a dispute and would like to settle it by the canonical fair and random process: flipping a coin.[1] If they were in the same room, they could just flip a coin, with (say) Alice winning the dispute in case of heads, and Bob winning in case of tails. However, they are thousands of miles apart, and they are only talking on the telephone. Furthermore, they do not have fancy modern phones with video cameras.

What can they do? One option is for Alice (say) to flip a coin, and she wins if the coin lands on heads and loses if the coin lands on tails. But this is not satisfactory from Bob's perspective: there is nothing to stop Alice from *claiming* that the coin landed on heads, regardless of the actual result. In fact, she could easily make such a claim without even flipping a coin at all.

Another thing they can try is for Bob to decide (secretly) whether he picks heads or tails, and after Alice announces the result of the coin flip, he reveals to her which he has chosen. But now we have the opposite problem: Bob can lie. If the coin lands on heads, then he can claim to have chosen heads, and if the coin lands on tails, then he can claim to have chosen tails. So, this scheme is also not satisfactory.

Instead, we need a way for both Alice and Bob to lock down their choices *before* a certain key piece of information is revealed. We do this by relying on a computationally difficult problem. Many such computationally difficult problems admit similar commitment schemes, but we will discuss one based on factoring.

Rather than flipping coins, Alice picks two large prime numbers, p and q, such that one of them is $1 \pmod 4$ and the other is $3 \pmod 4$. Alice computes $n = pq$ and reveals this number to Bob. However, she does not reveal p or q. Since $n \equiv 3 \pmod 4$, Bob can already verify that one of p and q is $1 \pmod 4$ and the other is $3 \pmod 4$. (Well, sort of: he cannot verify that n has only two prime factors. But more on that later.)

[1]They haven't learned about the hacks we discussed on page 177.

Figure 16.7 A combination padlock

The number n is Alice's part of the commitment scheme. Now, it's Bob's turn to do something. He knows that one of the prime factors of n is $1 \pmod 4$ and the other is $3 \pmod 4$, but he doesn't know whether it is the smaller or larger one that is $3 \pmod 4$. So, he makes a guess. That guess is Bob's part of the commitment scheme.

Now, Alice reveals the two numbers p and q, and both Alice and Bob can easily check whether Bob's guess was right or not. Furthermore, they can check whether Alice cheated or not. For example, Bob can verify that $pq = n$. He can also verify that p and q are both primes. (Remember that primality testing can be done in polynomial time, i.e., quickly!)

16.10 Problems

(1) For each congruence class $a \pmod p$, determine whether or not a is a square modulo p. For those that are squares, see if you can compute all the square roots. How many are there?

- (a) $26 \pmod{83}$
- (b) $14 \pmod{47}$
- (c) $9 \pmod{101}$
- (d) $45 \pmod{53}$

(2) Suppose that p and q are distinct primes, and that $n = pq$. Show that a is a perfect square modulo n if and only if a is a square modulo p and a square modulo q.

(3) Prove Theorem 16.4.

(4) Victor is colorblind. Peggy has two billiard balls, which are identical except for color: one is green and one is red. Peggy would like to convince Victor that

they are different, without telling him which one is which. Design a protocol for them to use.

(5) Victor has two combination padlocks like the one in Figure 16.7. Peggy claims that she knows the combination to one of them. Design a procedure that allows her to convince Victor that she knows the combination to one, without him learning what the combination is, or which one she knows the combination for.

(6) Peggy has two identical padlocks, and she claims that they both have the same combination. Design a procedure whereby she can convince Victor of this, without him learning the combination.

(7) Peggy claims to know the number of coins in a large jar. Come up with a protocol whereby she can convince Victor of this. (You may suppose that she can instantly determine the number of coins in the jar at all times.)

(8) (a) Given a deterministic oracle that, for every a and n, either returns some t so that $t^2 \equiv a \pmod{n}$ or promises that no such t exists. ("Deterministic" means that the oracle always returns the same answer given the same input.) Explain how to use this oracle to factor numbers quickly. (Hint: Vs n vf n fdhner zbqhyb a, ubj znal fdhner ebbgf qbrf n unir?)

(b) Given an oracle that returns the prime factorization of a number n, explain how to determine, given a and n, whether there is some t such that $t^2 \equiv a \pmod{n}$, and if such a t exists, to find an example. (You may assume that you know how to take square roots modulo p when they exist, even if this is actually false. At any rate, there *are* polynomial-time algorithms for doing that.)

(9) Let $G = (V_1, E_1)$ and $H = (V_2, E_2)$ be two graphs, so that E_i is a set of pairs of vertices in V_i, for $i = 1, 2$. We say that G and H are isomorphic if there is a bijective function $\sigma : V_1 \to V_2$ such that $\{v, w\} \in E_1$ if and only if $\{\sigma(v), \sigma(w)\} \in E_2$. (That is, G and H are "the same" up to relabeling the vertices.) Find a zero-knowledge proof for determining that G and H are *not* isomorphic. (You may assume that Peggy knows an actual isomorphism of the graph, and furthermore than she can solve the graph isomorphism problem for *every* pair of graphs.) This is interesting, because there does not seem to be a general way of producing a short proof that two graphs are not isomorphic, i.e., graph non-isomorphism is not believed to be an NP problem. By contrast, one can produce a short proof that two graphs *are* isomorphic by simply writing down an isomorphism.

(10) Find a zero-knowledge proof for showing that two graphs *are* isomorphic.

(11) Consider the following attempt at a commitment scheme. Alice chooses a prime p, a primitive root g modulo p, a number x, and computes $h \equiv g^x \pmod{p}$. She then tells Bob p, g, and h, and asks Bob whether x is even or odd. Why is this a bad idea?

(12) In the previous problem, suppose that Bob is only supposed to have a one-third chance of winning the commitment, and Alice asks him to guess $x \pmod{3}$. Is this a good commitment scheme?

(13) Find a commitment scheme for a coin flip based on the discrete logarithm problem.

(14) Modify the factorization-based commitment scheme to the case where Bob is supposed to have an r/s chance of winning, where r/s is a rational number in lowest terms.

(15) Find a different zero-knowledge proof for Sudoku.

(16) Find a zero-knowledge proof for solving Rubik's cubes.

(17) A graph G is said to have a Hamiltonian circuit if there is a sequence $v_1, v_2, \ldots, v_n, v_1$ of vertices, starting and ending with the same vertex, such that

  (a) each vertex of G is on the list $v_1, \ldots, v_n$ exactly once,
  (b) for each i, there is an edge between v_i and v_{i+1}, and also an edge between v_n and v_1.

  Determining whether G has a Hamiltonian circuit as an NP-complete problem. Find a zero-knowledge proof for determining whether G has a Hamiltonian circuit. (You may assume that Victor cannot solve the graph isomorphism problem.)

(18) Come up with a zero-knowledge proof for graph 3-colorability that Peggy and Victor can enact if Victor believes that Peggy has a time machine.

Chapter 17
Secret Sharing, Visual Cryptography, and Voting

17.1 Secret Sharing

Suppose you have n people, with whom you wish to share a secret. However, you do not wish to entrust any of them *individually* with the secret. Rather, you only want to allow them to learn what the secret is when a significant coalition of the people, at least k, say, work together to learn it. Furthermore, you don't want to allow smaller coalitions to learn bits of the secret; you want any coalition of at most $k - 1$ people not to be able to learn anything at all. How do you do it?

First, the trivial case: if $k = 1$, then you just tell each of the n people the secret.

The case of $k = n$ is also relatively straightforward. For example, suppose that the secret s is an integer, or an element of $\mathbb{Z}/m\mathbb{Z}$. (They both work the same way.) Choose numbers $a_1, \ldots, a_n$ so that $a_1 + a_2 + \cdots + a_n = s$. Then, when they all work together, they can reconstruct s. However, if a single person refuses to help, then the remaining $n - 1$ people can learn nothing. (The remaining number could be any integer, positive or negative, or any element of $\mathbb{Z}/m\mathbb{Z}$.)

If $k = 2$, the problem is already interesting: how do you arrange that no individual person can work out the secret, while any pair of people can?

Based on the solution for $k = n$, there is a stupid solution: for any two-element subset $\{i, j\}$ of $\{1, 2, \ldots, n\}$, create a pair of numbers $a_{i,j,1}$ and $a_{i,j,2}$ such that $a_{i,j,1} + a_{i,j,2} = s$. Give person i the number $a_{i,j,1}$ and person j the number $a_{i,j,2}$. Repeat for all pairs i, j. This requires choosing $\binom{n}{2} = \frac{n(n-1)}{2}$ pairs of numbers that sum to s, and each of the n people has to keep track of $n - 1$ numbers, to take care of all possible coalitions.

In fact, this scheme extends much more generally: you can control exactly which coalitions can work out the secret. Suppose you have a collection of coalitions you wish to allow: T is a collection of subsets of $\{1, \ldots, n\}$, the coalitions that are allowed to work out the secret. T should satisfy the property that if $U \in T$ and $U \subset V$, then $V \in T$ as well. (Coalition V can just ignore the information of the people not in coalition U.) To each $U \in T$, construct a set of $|U|$ numbers that sums to s. Then distribute those $|U|$ numbers to the members of U. In fact, we only need to do this for the *minimal* allowable coalitions: those that do not contain any smaller allowable coalitions.

If the number of allowable coalitions is small, this strategy may be viable. But it quickly gets out of hand: if there are 1000 people, and you want a coalition of 500 of them to be able to access the secret, then you need $\binom{1000}{500} \approx 2.7 \times 10^{299}$ sets of numbers summing to s. That is far too many.

17.2 Shamir's Secret-Sharing Scheme

The idea behind Shamir's secret-sharing scheme is to encode bits of the secret as the values of a polynomial. The key observation is that it takes k points to determine a polynomial of degree $k - 1$: two points determine a line, three points determine a parabola, and so forth.

So, again, we assume that s is an element of $\mathbb{F}_p$ (for some large p). Choose random $c_1, \ldots, c_{k-1} \in \mathbb{F}_p$, and define

$$f(x) = s + c_1x + c_2x^2 + \cdots + c_{k-1}x^{k-1}.$$

Observe that the secret is $f(0) = s$. Also observe that c_{k-1} *might* be 0, which would make the polynomial have degree strictly less than $k - 1$; this does not cause any problems.

Now, we must distribute the numbers $a_1, \ldots, a_n$ to the n people. We set $a_i = f(i)$, and give person i the number a_i. Any k people can reconstruct f, since k points suffice to determine a polynomial of degree at most $k - 1$. But $k - 1$ people cannot. (In fact, they can say nothing at all about the secret; see problem 6.)

It is worth discussing exactly *how* to reconstruct a polynomial of degree at most $k - 1$ from knowing k values. It's easy: there's a formula.

Theorem 17.1 (Lagrange Interpolation Formula) *Suppose we have points* (x_1, y_1), $\ldots, (x_k, y_k)$, *with all the* x_i*'s distinct. Then, there is a unique polynomial* $f(x)$ *of degree* $\leq k - 1$ *interpolating these points, i.e.,* $f(x_i) = y_i$ *for all* i. *This polynomial is*

$$f(x) = \sum_{j=1}^{k} y_j \prod_{\substack{1 \leq m \leq k \\ m \neq j}} \frac{x - x_m}{x_j - x_m}.$$

So, we just find f by using the Lagrange Interpolation Formula.

Just kidding! Okay, the formula is fine if you're writing a computer program to do this. But there's a method of doing the interpolation that is very easy and doesn't require memorizing anything. If you are a human, I recommend you use that method instead.

We start by doing an easier interpolation problem, interpolating the points (x_1, y_1) and $(x_2, 0), (x_3, 0), \ldots, (x_k, 0)$. This is easy to do: any polynomial passing through the points $(x_2, 0), \ldots, (x_k, 0)$ is of the form $c(x - x_2)(x - x_3) \cdots (x - x_k)$. So, choose c to make it interpolate the point (x_1, y_1). Call this polynomial $f_1(x)$.

Similarly, we can find a polynomial interpolating the points (x_2, y_2) and $(x_1, 0)$, $(x_3, 0)$, $(x_4, 0), \ldots, (x_k, 0)$, using exactly the same method. Call this polynomial $f_2(x)$. Similarly, construct $f_3(x), \ldots, f_k(x)$.

Finally, we put them together: $f(x) = \sum_{j=1}^{k} f_j(x)$. Since $f_1(x_1) = y_1$ and $f_2(x_1) = \cdots = f_k(x_1) = 0$, we have $f(x_1) = y_1$. Similarly, $f(x_j) = y_j$ for all $1 \le j \le k$.

The Lagrange Interpolation Formula is doing the same thing as we just did, but put into the form of a concise and non-illuminating formula.

Example Let us find a polynomial of degree at most 4 passing through the points $(0, 2), (1, 4), (2, 7), (4, 6)$, and $(8, 19)$ over $\mathbb{F}_{23}$. We start by finding $f_1(x)$, a degree-4 polynomial passing through $(0, 2)$, $(1, 0)$, $(2, 0)$, $(4, 0)$, and $(8, 0)$. To make it pass through the last four points, we take $f_1(x) = c(x-1)(x-2)(x-4)(x-8)$, and then we have to find c. We have $2 = f_1(0) = c(-1)(-2)(-4)(-8)$, so $c = 32^{-1} = 18$. Thus $f_1(x) = 18(x-1)(x-2)(x-4)(x-8)$.

Next, we compute $f_2(x)$, which passes through $(0, 0)$, $(1, 4)$, $(2, 0)$, $(4, 0)$, and $(8, 0)$. This one has the form $f_2(x) = cx(x-2)(x-4)(x-8)$, and plugging in $x = 1$, we get $4 = f_2(1) = c(1)(-1)(-3)(-7)$, so $c = -4 \times 21^{-1} = -4 \times 11 = 2$, so that $f_2(x) = 2x(x-2)(x-4)(x-8)$. Continuing on in this way, we find that $f_3(x) = 7x(x-1)(x-4)(x-8)$, $f_4(x) = 10x(x-1)(x-2)(x-8)$, and $f_5(x) = 18x(x-1)(x-2)(x-4)$. Putting this all together, we have

$$f(x) = \sum_{i=1}^{5} f_i(x) = 9x^4 + 19x^3 + 7x^2 + 13x + 2.$$

We can easily check that this polynomial does indeed interpolate the five points we specified at the beginning.

17.3 Blakley's Secret-Sharing Scheme

Around the same time that Shamir developed his secret-sharing scheme based on polynomials, Blakley came up with a different method, based on hyperplanes, or linear algebra if you prefer. Suppose we have n people, and we wish to allow any coalition of k people to work out the secret $s \in \mathbb{F}_p$. To do this, we construct n linear equations in k variables over $\mathbb{F}_p$.

To do this, we fix not-necessarily-distinct elements $c_1, \ldots, c_k \in \mathbb{F}_p$, with $c_1 = s$. Then we find n linear equations in k variables $x_1, \ldots, x_k$ with coefficients in $\mathbb{F}_p$, so that setting $x_j = c_j$ for $1 \le j \le k$ satisfies each equation. More precisely: we pick $a_{11}, a_{12}, \ldots, a_{nk}, b_1, \ldots, b_n \in \mathbb{F}_p$ randomly, subject to the constraint that, for each i,

$$a_{i1}c_1 + a_{i2}c_2 + \cdots + a_{ik}c_k = b_i.$$

We then give the ith equation to person i.

Well, we can't do this *completely* randomly, but most choices are okay, at least if p is large. We need to make sure of a few things. First, it is no good if, say, one of the equations is $a_{i1}x_1 = b_i$, because then person i can determine what a solution for x_1 singlehandedly. Similarly, we need to make sure that no coalition of $k-1$ people can determine the unique solution c_1 for x_1. We also need to make sure that any coalition of k people *can* determine c_1.

17.4 Visual Cryptography

Let's go low-tech. Most of the time in cryptography, we wish to encrypt words or numbers (or perhaps some more exotic thing, such as points on elliptic curves). But what if we want to encrypt a picture? This question was first investigated by Moni Naor and Adi Shamir in [NS95].

One way of doing this is to save the picture in some digital format such as a jpg file, which then converts the picture to a string, which can convert to a number and encrypt as usual.

But isn't it more *fun* to encrypt *the picture itself*? Let's not worry just yet about the usual cryptographic problem of establishing a secure key over a public channel and instead just think about what it could possibly mean to encrypt a picture.

For us, a picture is something that we can recognize with our bare eyes, without technological enhancements. Now, our eyes aren't perfect; we don't see things that are really small, and we can't detect very subtle differences between two pictures. For our purposes, this will turn out to be a feature, not a bug!

Let us suppose that the picture we wish to share is in black and white only. (It is possible to modify our protocol to be able to handle color images as well, but it adds unnecessary extra complications.) When we see a region that is mostly white, with a few black dots in it, then it is easy for our eyes to interpret the region as being completely white. Similarly, when we see a region that is mostly black with a few white dots in it, we interpret it as being completely black.

So, this gives us an idea. Suppose we wish to encrypt a picture, say of the letter A, as in Figure 17.1, left. We first pixelate it; ideally this would be done in more detail, but let us keep it severely pixelated for ease of understanding. Let us also invert it, so that the letter is in white and the background is in black; see Figure 17.1, right.

Next, what we do is to subdivide each pixel into a 2×2 grid of subpixels. We replace every black pixel with a 2×2 grid of black subpixels, while we replace every white pixel with one of the two arrangements of subpixels in Figure 17.2, chosen at random.

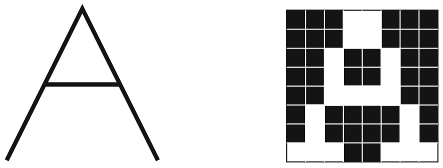


Figure 17.1 Left: An A; Right: A pixelated A

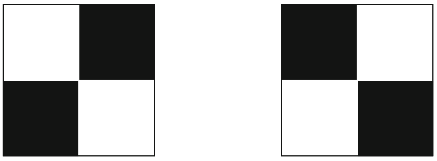


Figure 17.2 The two possible 2 × 2 arrangements of sub-pixels that code for white pixels

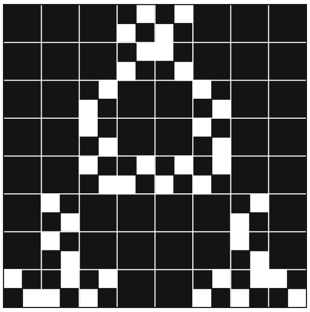


Figure 17.3 The letter A, pixelated and subdivided

Our letter A might then look like Figure 17.3. As we can see, it is still quite easy to read, although it helps to enlarge it a bit, as we have done.

Now, to turn this letter into something cryptographic, we imagine superimposing two transparencies, one on top of the other, in order to generate the image in Figure 17.3. Because of how physical reality works, when we overlay two transparencies, then we see a black pixel unless *both* of the pixels overlaid are white. We want to choose our two layers so that each pixel on each layer looks like one of the two possibilities in Figure 17.2. For any given pixel, if both layers are the same, then we end up with one of the possibilities in Figure 17.2; if the two are different, then we end up with a fully black pixel.

Now, this suggests a way of encoding the picture. We first select a key, which is a random collection of 2 × 2 subpixels as in Figure 17.2. The "plaintext" is the original picture, pixelated but not further subpixelated. Now, we create the ciphertext. It has the same form as the key, except that it's not random. In every position where the original image had a black pixel, we put a 2 × 2 arrangement of sub-pixels that *differs* from that of the key, whereas for every white pixel, we use the *same* arrangement. This way, when we overlay the two images, every black pixel is fully black, and every white pixel is half-white, half-black as in Figure 17.2.

Observe that an eavesdropper who only has access to one transparency learns *exactly* nothing: any pixel could still be either black or white. The key functions as a one-time pad for this cryptosystem.

As for secure key generation, this is not so difficult, at least, given access to a computer. (And, realistically, how are you supposed to create these transparencies without a computer anyway?) The two parties agree on a key, say via Diffie–Hellman, and then convert the key into a string of 0's and 1's; every 0 is transformed into the sub-pixel arrangement on the left of Figure 17.2, whereas every 1 is transformed into the arrangement on the right.

In their original paper on visual cryptography, Naor and Shamir also discuss how to do a secret-sharing version, where n people get shares of the key, and any coalition of k people (but no coalitions of $k-1$ people) can view the picture.

Remark 17.2 The idea of combining two (or more) uninteresting patterns into one meaningful pattern is of interest to people other than cryptographers as well. In art, this is the concept of the *Moiré pattern*. It turns out that it is possible to make all sorts of interesting patterns by taking two copies of the same pattern, such as a bunch of parallel lines, and misaligning them slightly. Much of the theory has been carefully worked out; see, for instance [Gab07].

17.5 Cryptographic Voting

Let us now put a bunch of the things we have studied together, so that we can create a cryptographic voting scheme. As Stalin pointed out,

> *The people who cast the votes decide nothing. The people who count the votes decide everything.*

The problem is that it is difficult to establish a transparent system: for example, what stops a dishonest vote counter from forging the results? Similarly, there are other serious problems that we must overcome in order to create a satisfactory voting system.

This is just the sort of setup where cryptographic protocols tend to thrive. The goal is to set up a protocol whereby voters can verify that their votes were counted, and furthermore that they were not tampered with.

More precisely, here are some of the things we expect out of a good voting protocol:

Democratic: Only eligible voters can vote, and each eligible voter can only cast ≤ 1 ballot.

Private: No one can tell how an individual voter voted. Being able to publish a list of voters who voted is allowable, but it must not be possible to determine from this list what any individual voter's vote was.

Receipt-free: Voters must not be able to get certificates verifying how they voted. More generally, they must *not* be able to prove to anyone else that they voted as claimed. The reason we want this is to ensure that there is no reliable way of buying or selling votes.

Accurate: The system must correctly tally the number of votes cast for each candidate.

No partial results: No one can learn any partial results before the voting period closes.

Individual verifiability: Voters must be able to verify that their votes were counted correctly.

Universal verifiability: Anyone can verify that the total is correct.

Cryptographic voting is quite a tough problem, because of the large number of possible adversaries. Most obviously, the vote *counters* are potential adversaries: we need a way of certifying that they count and report the votes correctly, and we must also ensure they cannot deanonymize the individual votes. On top of that, individual voters are potential adversaries: they can try to sell votes, for example. They might also try to do other similar shady things: maybe Bob is uninterested in the election and just wants to copy Alice's ballot. We don't want to allow him to do that. Alternatively, maybe he wants to cancel Alice's ballot. We don't want to allow him to do that either. (All of these are potential forms of coercion, which are bad for democracy.)

In addition, some of the demands appear to be at odds with each other. In particular, there is an apparent conflict between the requirement that ballots be receipt-free, and the requirement of individual verifiability: how can I determine if my vote was counted if I don't get to keep any documentation of it? Maybe I can memorize my ballot and a serial number, but it is bad policy to require voters to memorize things in order to verify that their votes are counted: most people would struggle to memorize a 9- or 10-digit number on command.

In general, it is safe to assume that *most* people are good. But any involved party *might* be compromised. Thus, we need to make the entire system secure against one corrupt party, or a small committee of corrupt parties. If everyone involved is corrupt, then the system may be compromised, but this is unlikely: by assigning parties with divergent interests different roles in the process, we can minimize the chances that everyone is compromised. As we have learned in the US, the Republicans and Democrats are incapable of working together, so if we assign them both roles in the process, then they will be unable to collude to corrupt the election results.

17.6 An Outline of a Cryptographic Voting Protocol

So, how does it work? As we mentioned before, it's complicated, and it uses much of the stuff we have already learned. We begin by giving an outline of the entire process. We will then fill in many of the details. So, here is, roughly, the idea of a system. This voting protocol is a modified version of the schemes proposed by Chaum and Neff. There are probably some loopholes here that make the complicated details in their schemes necessary, but we leave that discussion for a more thorough study of cryptographic implementations than is presented here.

(1) First, an encryption key is created. We do not want any party to have full access to the key, so this process is done in a distributed manner: each of several parties has a partial key.
(2) Next, each eligible voter is assigned a unique voter ID token. The voter ID token is required in order to log onto the system to vote. The token may not be reused.
(3) Next, the voters must submit their votes, which then get encrypted. Typically, we use ElGamal encryption for this, because it is easy to encrypt or decrypt a message in small stages using ElGamal, as we shall soon see.

(4) Each voter confirms that eir vote was cast as intended.
(5) Votes are decrypted. This is done in stages, with each partial key holder doing a small part of the decryption, and then verifying to an external auditor that the votes were actually decrypted correctly. Special care must be taken for the final decryption.
(6) After every stage of the decryption, the increasingly decrypted ballots are posted publicly, so that auditors can verify their accuracy.

17.7 Key Generation

In order to generate the key, we use ElGamal key generation, either in its standard formulation in $\mathbb{F}_p^\times$ for a large prime p, or in its elliptic curve formulation. In what follows, we assume that we are working with the $\mathbb{F}_p^\times$-version of ElGamal, but it is easy to modify it to work for elliptic curve ElGamal.

We do not want to entrust the entire private key to a single party, so we allow several parties to create their own partial keys, combining them to create a full key. First, we fix, once and for all, a large prime p and a primitive root g modulo p. These are public information, available to the key generators and everyone else.

Let us suppose that we have t partial key generators. For each i with $1 \le i \le t$, the ith key generator picks a random a_i and computes $h_i = g^{a_i} \pmod{p}$, which then becomes public. The full public key is then

$$h \equiv \prod_{i=1}^{t} h_i \pmod{p} \equiv g^{\sum a_i} \pmod{p}.$$

Note that everyone can compute the full public key, even though no individual knows the private key, but only a small share of it.

Each voter will encrypt eir vote "in one go," using the full h as the encryption key, but decryption will have to be done in stages: first, key generator 1 will do a partial decryption, then key generator 2 will do a partial decryption, and so forth, and then key generator t will do the final decryption.

However, we have to be a little bit careful with person t, since ey could simply choose to withhold the results of the election if the results are not to eir liking. There are several solutions to this problem.

One solution is to use secret sharing for this final piece: we share the tth piece of the private key a_t among many people, so that any committee of size k can determine the value of a_t, for some suitable k. We might even want to do this without having a dealer to give out the shares, since that entrusts too much power to a single person, who could give out fake shares and hold up the election results if desired. See problem 13 for more on how to do secret sharing without a dealer. In fact, we could do this with all the pieces.

17.8 Assigning Tokens

Each eligible voter probably has some unique piece of identifying information, such as a passport. (This isn't quite right, because some citizens do not have passports. But we can imagine the voting authority providing some similar documentation to all registered voters.)

When a voter goes to vote, ey must scan eir passport or other document in a machine. Now, we don't want a vote to be traceable, so the machine should not just store the passport number directly; instead, it should *hash* the passport number. To do this, we apply some very complicated function $f : \{\text{relatively small numbers}\} \to S$, where S is a very large set. We want this function f, called a *cryptographic hash function*, to have the following two properties:

Collision resistance: There should not be two different passport numbers m and n such that $f(m) = f(n)$.

One-way: Given $s \in S$, it should be very difficult to find an n such that $f(n) = s$, or even to determine whether one exists.

Almost any "sufficiently complicated" f is a cryptographic hash function, if S is extremely large. Recall that we briefly mentioned cryptographic hash functions in Section 12.6, when we discussed digital signatures.

Ideally, we want the token to go through several rounds of hashing on different machines. The reason is that any hashing machine could be corrupted. What it means for a hashing machine to be corrupted is that it stores the results of all the hashes as it performs them, and then possibly sells the results to a third party, who then knows something about the votes. But if we have several rounds of hashing, then it is unlikely that *every* hashing machine is corrupted, and it only takes one honest hashing machine for the votes to be anonymous.

Now, if a voter tries to vote a second time, then the hash machines will notice that: the hash functions are completely deterministic, so the same voter ID gets hashed the same way both times. The machines notice that the hashed voter ID has already been used, and the voter is denied the chance to vote a second time.

17.9 Encrypting and Verifying the Votes

Now that we have authenticated the voters, we allow them to vote. Each voter prepares a ballot with eir vote. There are then two things that need to be done:

(1) The voter must check that the vote was cast correctly.
(2) The vote must be encrypted for future use.

In fact, we combine these two things into one: the encryption essentially *is* a verification for the voter. One possibility, of course, is just to have the voter do some ElGamal-style computation to encrypt the vote, then remember the number. But this is unsatisfactory: we can't expect humans to do lengthy computations to verify that their votes were cast as intended. (This is especially problematical for professional mathematicians, since part of our training involves forgetting how to do arithmetic.)

Instead, we want a method of encryption that can be verified *at a glance*. This is the realm of visual cryptography! After the voter submits eir ballot on a machine, the machine produces two transparencies with lots of dots on them, so that when they are superimposed, the ballot appears as cast. Now, this doesn't really sound like encryption. However, the machine uses the ElGamal public key h in order to construct the two transparencies. We won't discuss further how that process works, but instead refer the reader to David Chaum's excellent paper [Cha04] on the topic. The voter then chooses one of the transparencies to serve as eir ballot; after decryption, either one is sufficient to serve as a ballot. The voter takes that transparency home, and the other one is shredded.

Remark 17.3 Several questions immediately arise. First, why can't the voter keep *both* transparencies? Second, why does the voter have to be able to *choose* which transparency to take, rather than (say) always getting the top one? Finally, doesn't the transparency count as a receipt of a vote?

Let's answer these questions one by one. First, we don't want to allow the voter to keep both pieces, because that would genuinely be a receipt: combine both transparencies to see a vote. So, this would lead to a coercible voting system: someone can pay me to vote in a certain way, and I receive the payment upon giving this other person my transparencies. That is bad, and this is the reason that polling places today do not provide any receipts after voting. However, refusing to provide receipts comes with drawbacks of its own: voters have to *trust* that their votes were counted (and were counted correctly). See the Stalin quote above.

Now for the second question: why do we have to allow the voter to choose? Well, suppose you have any black and white pixelated picture whatsoever (of a given size), and you wish to split it into two layers of transparencies, as described in Section 17.4. Then you can make the top layer (or the bottom layer, if you prefer) be *anything at all*, and then choose the second layer so that the combined picture comes out right. So, we have to worry about the possibility of the transparency machine cheating: it can cheat on one of the layers, but then the other layer is forced. So, the machine could get away with cheating (and creating one layer of transparency that specifies a different vote), but only half the time, namely if the voter picks the cheating transparency layer. So, the voting machine can only get away with this 50% of the time. When this happens, the other layer is probably gibberish when decrypted. As a result, if the voting machine is cheating on a large scale, then it is extremely likely that someone will notice once the votes are decrypted, and the election will be declared a fraud.

Finally, isn't the transparency a receipt? In theory, it is possible to decrypt the transparency into a vote, which is what the decryption process will eventually do. But without the help of all the key generators, it will not to be possible to convert it back into the plaintext vote. So, this receipt cannot be reliably used for coercion, in the sense of guaranteeing that the voter voted in a specific way. On the other hand, it *can* be used as a receipt for the voter's sake of verification: the voter can match the receipt to a bulletin board containing all the (encrypted) votes and see that it shows up there.

After all this is done, all the encrypted ballots (the dot patterns that each voter chose) are put on a public bulletin board (presumably on the Internet), for all to investigate. They're encrypted, so no one can tell what they say. But anyone who wishes can use this information to verify that all future steps are performed correctly. For example, when a voter returns home from the polling place, ey can verify that eir ballot is on the bulletin board. We assume that there is sufficient randomization in the encryption process that, even if two people cast identical ballots, their encrypted ballots end up looking different.

17.10 The Decryption Mix-Net

Now that all the encrypted ballots have been placed on the bulletin board, it's time to decrypt them. Okay, the ballots are in some weird visual form, but from now on, we won't worry about that; instead, we'll just treat them as elements of $\mathbb{F}_p^\times$, as usual for ElGamal encryption.

Recall that no one has the full key, but the t key generators each have a portion of the key. So, each one in turn has to peel off a layer of the encryption, until they're all gone. However, it's not so good if they simply decrypt the messages "in place," since then a third party can match up the encrypted ballot with the decrypted one, and if that person knows where a voter V's ballot was originally, then that person can determine how V voted. We don't want that.

So, after each layer of decryption, the decrypter must also scramble the ballots. Let us suppose that there are n voters, and that their fully encrypted ciphertexts are $C_{1,0}, C_{2,0}, \ldots, C_{n,0}$. Then, decrypter number 1 must decrypt each of the $C_{i,0}$'s and scramble them, to obtain $C_{1,1}, C_{2,1}, \ldots, C_{n,1}$. Note that $C_{i,1}$ is not necessarily the decryption of $C_{i,0}$; rather, it is the decryption of *some* $C_{j,0}$, and no one else should know which.

In order to preserve anonymity, we expect the decrypters not to disclose any information about the permutation used. This immediately presents a few problems:

(1) How do we *know* that the decrypters are not selling their permutations to a third party?
(2) How do we verify that the votes were correctly decrypted and scrambled?

The first question is outside the realm of cryptography: we have no way of checking that the decrypters did *not* sell the permutations. But this is why we have many decrypters: if a third party is trying to deanonymize the votes, then this is only possible if ey can collect *all* the permutations used by *all* the decrypters. As long as at least one of the decrypters is honest, the whole scheme is safe.

The second question, on the other hand, is very much within the realm of cryptography: how do we verify that the votes were decrypted and scrambled correctly?

At each stage r, we must show that the ballots $C_{1,r-1}, \dots, C_{n,r-1}$ are an encryption of $C_{1,r}, \dots, C_{n,r}$. We want to do so, without the decrypter having to give away any information about the permutation, or at least as little as possible. This is the perfect place to invoke a zero-knowledge proof. We'll get there in several steps.

Let us first forget about the permutation part and just pretend there is only one ballot to be encrypted. So, we want to show that $C_{1,r-1}$ is an ElGamal encryption of $C_{1,r}$. The public key is (p, g, h), where $h_r \equiv g^{a_r} \pmod{p}$, and a_r is secret; for simplicity we'll write a instead of a_r. If done correctly, we have $C_{1,r} = (g^x, mh_{r+1}^x \cdots h_t^x)$ and $C_{1,r-1} = (g^x, mh_r^x h_{r+1}^x \cdots h_t^x)$, for some x. Let us write $C_{1,r} = (\alpha_1, \beta_1)$ and $C_{1,r-1} = (\alpha_2, \beta_2)$. Then consider the quadruple $(\pi, \rho, \sigma, \tau) = (g, h_r, \alpha_1, \beta_1^{-1}\beta_2)$. If done correctly, then $\sigma = g^x$ and $\tau = (mh_{r+1}^x \cdots h_t^x)^{-1}(mh_r^x h_{r+1}^x \cdots h_t^x) = h_r^x$, so we should have $(\pi, \rho, \sigma, \tau) = (g, h_r, g^x, h_r^x)$. Note that, since $h_r = g^a$, we can also write this quadruple as $(\pi, \rho, \sigma, \tau) = (g, g^a, g^x, g^{ax})$. We need to verify that this is correct.

This problem is known as the *Decision Diffie–Hellman* (DDH) problem, since this amounts to determining whether a solution to the Diffie–Hellman problem (that is already handed to you, so you don't have to figure it out from scratch) is correct. It appears to be computationally unfeasible to solve the DDH problem. So instead, the encrypter must give a zero-knowledge proof that it was done correctly.

One of the ways to give a zero-knowledge proof for the DDH problem is the *Chaum–Pederson Protocol*. It works as follows: The prover wants to convince the verifier that $(\pi, \rho, \sigma, \tau) = (g, g^a, g^x, g^{ax})$ is a valid Diffie–Hellman solution. First, the prover picks some s (and keeps it secret), and then computes and sends π^s and ρ^s to the verifier. The verifier then picks and sends a challenge c. The prover then sends $t = s + xc$. The verifier accepts if and only if $\pi^t \equiv g^s \times \sigma^c \pmod{p}$ and also $\rho^t \equiv \rho^s \times \tau^c \pmod{p}$. Note that, while s is unknown, g^s and σ^s are known to the verifier.

So, using the Chaum–Pederson Protocol, the mixer can demonstrate to any interested third party that *one* ballot was encrypted correctly. But we really need the mixer to prove that *many* ballots were encrypted correctly. Now, there are ways of doing this, but they are quite complicated; see, for instance, [Nef03]. Instead, after round i of decryption (i.e., decrypting the $C_{i,r-1}$'s into $C_{j,r}$'s), an independent verifier or a team of independent verifiers points to one of the $C_{i,r-1}$'s or $C_{j,r}$'s, and the decrypter must point to the corresponding ballot in the other layer. Then, they perform a Chaum–Pederson test to convince the verifier that the decrypter is telling the truth. They do this several times, until the verifier is satisfied.

Now, if the decrypter is cheating and tossed in some new ballots and deleted some others, then this *might* not be detected, but it's also very risky: it might happen that the verifier asks for the decryption of a ballot that was deleted, or for the encryption of an added ballot. In that case, the deception will immediately be noticed.

However, we do want to be a little bit careful in this process, in order to protect voter anonymity. It's no good if we allow the verifier to query *all* the votes, because that gives away the shuffle. Furthermore, the verifier can't keep tracing the same

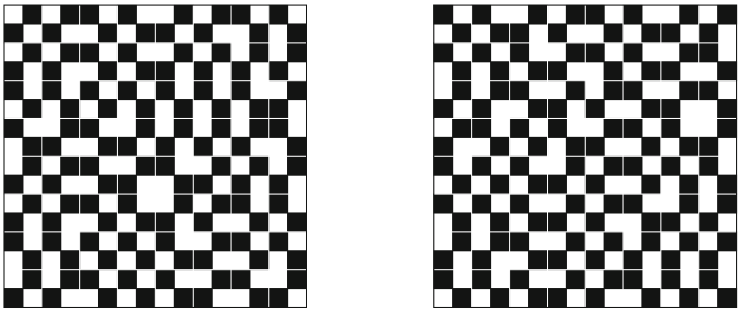


Figure 17.4 Two shares of a visual secret

ballots each round, or else that voter's anonymity has been destroyed. Thus, we require that the verifier make *random* queries.

As we mentioned before, the final decrypter has a lot of power to hold up the election, by refusing to release the results if they are not desirable. (In fact, given that there is likely to be a lot of pre-election polling, any of the decrypters could do this.)

To deal with this, we can make each decrypter have just a piece of a shared secret (ideally with no dealer), as mentioned earlier. But also, one nice feature of ElGamal encryption is the decryptions can be done in any order, and we still recover the same plaintext message. Originally, we planned to have decrypter 1 go first, then decrypter 2, then decrypter 3, and so forth. But we could have several parallel rounds. In the first round, all the decrypters decrypt the votes with their only private keys and scramble the ballots, doing their Chaum–Pederson verifications. Next, they pass their partially decrypted ballots on, with decrypter 1 passing the messages to decrypter 2, decrypter 2 passing the messages to decrypter 3, and so forth. After all t rounds of this, all the decrypters should have the fully decrypted ballots. This also allows for double-checking: if they don't all have the same results, then something went wrong somewhere.

17.11 Problems

(1) n people would like to determine their average annual income. However, no one wants to disclose eir income to anyone else, or anything about it (other than that it's $\leq$ the sum of all the incomes). How can they do it? You may assume that the value of n is known.

(2) Find a polynomial $f(x)$ of degree at most 4 over $\mathbb{F}_{13}$ such that $f(2) = 9$, $f(3) = 12$, $f(4) = 6$, $f(8) = 10$, and $f(11) = 6$.

(3) Suppose $a_0, a_1, \ldots, a_{p-1}$ are arbitrary (not necessarily distinct) elements of $\mathbb{F}_p$. Is there a polynomial $f(x)$ with coefficients in $\mathbb{F}_p$ such that $f(n) = a_n$ for

$0 \le n \le p-1$? If so, how many such polynomials are there? How small can you make its degree, in the worst case?

(4) A polynomial $f(x)$ with coefficients in $\mathbb{F}_p$ is said to be a *permutation polynomial* if, for every $b \in \mathbb{F}_p$, there is some $a \in \mathbb{F}_p$ so that $f(a) \equiv b \pmod{p}$. (That is, f permutes the values of $\mathbb{F}_p$.) How many permutation polynomials are there, with coefficients in $\mathbb{F}_p$ and degree d?

(5) Let F be a field, and let $f : F \to F$ be a function. Define $\Delta f(x) = f(x+1) - f(x)$, and $\Delta^n(x) = \Delta(\Delta^{n-1}(x))$ for $n \ge 2$.

(a) Show that if f is a polynomial, then for each n, $\Delta^n(x)$ is a polynomial in x.
(b) Show that if f is a polynomial of degree $\le n-1$, then $\Delta^n f(x) = 0$ for all $x \in F$.
(c) If F is an *infinite* field, show that if $\Delta^n f(x) = 0$ for all $x \in F$, then f is a polynomial of degree $\le n-1$.
(d) Find a counterexample if F is a finite field.

(6) (a) Suppose that we are using Shamir's secret-sharing scheme. Show that if $k-1$ people try to form a coalition to determine the secret, then they learn nothing. That is, for each $t \in \mathbb{F}_p$, the probability that $s = t$ given their pieces of information is $\frac{1}{p}$.
(b) Show that a coalition of $k-1$ people *might* be able to learn *something* about the secret if we tried to use Shamir's secret-sharing scheme with $s, c_i, a_i \in \mathbb{Z}/n\mathbb{Z}$ if n is composite. What if $s, c_i, a_i \in \mathbb{Z}$?

(7) The images in Figure 17.4 are two shares of an 8×8 visual secret. Find the secret.

(8) Verify that the Chaum–Pederson Protocol is a zero-knowledge proof.

(9) Describe, in detail, a secret-sharing scheme based on the Chinese Remainder Theorem. How is the secret encoded? Verify that no ineligible coalition can gain any partial knowledge of the secret using your scheme. Note that the Shamir and Blakley schemes require that any (minimal) coalition capable of recovering the secret has the same size. Explain how a Chinese Remainder Theorem-based scheme need not require this.

(10) Think of other secret-sharing schemes.

(11) In terms of n, k, and p, how many ways are there of choosing $a_{11}, \ldots, a_{nk}$ so as to satisfy the requirements of the Blakley secret-sharing scheme?

(12) Three cryptographers are having dinner at a restaurant. When the time comes to pay the bill, the waiter informs them that someone has already paid their bill. They would like to determine whether the donor is one of them, or someone else. However, if one of the cryptographers paid, then the others would like to respect that person's anonymity. Devise a scheme whereby they can determine whether someone from outside paid their bill or not.[1]

(13) Typically, a secret-sharing scheme requires a trusted party to distribute the secret. Find a modification of the Shamir secret-sharing scheme that allows the

[1]This problem is called the *Dining Cryptographer Problem*. It was proposed by David Chaum in [Cha88].

n parties to come up with a secret on their own, so that any coalition of k can unlock the secret, without having a trusted party to distribute the secret.[2]

(14) Understand why every step in our cryptographic voting protocol is necessary. Can you find loopholes in our scheme that may undermine its security, verifiability, anonymity, or other desirable properties?

(15) What other desirable properties might we want in a cryptographic voting protocol?

[2]See [Ped91] for one solution.

Chapter 18
Quantum Computing and Quantum Cryptography

In this chapter, we will look at how cryptography can work using quantum computers. Both cryptography and cryptanalysis behave completely differently in the quantum world from in the classical world. Classical cryptosystems such as RSA and ElGamal can be broken easily using a quantum computer, so in this respect quantum computers help eavesdroppers and attackers. On the other hand, there are new cryptosystems in a quantum world that are *provably* unbreakable, because quantum mechanics allows us to create a one-time pad, even in the presence of an eavesdropper.

Taking a thorough look at quantum computation would require an entire book in itself, and there are several such books available, such as [KLM07] and [NC00]. Doing things in the most thorough manner would require considerable background in linear algebra, which we have not assumed in this book, and also at least a bit of background in quantum mechanics, although this part can be de-emphasized for someone willing to treat the subject more as a formal game than as a representation of reality. We shall only give a cursory overview, so that you get some idea of how it works. It is also important to remember that, at the moment, quantum computers do not exist, and we do not know when, if ever, they will be built. So, this entire chapter is purely theoretical for the time being.

18.1 Quantum Versus Classical Computers

A classical computer is based on the notion of a *bit* (short for *binary digit*). This is the smallest possible amount of information: it is a single number, which is either 0 or 1. Classical computers express everything in terms of bits. For example, an arbitrary number can be represented as a sequence of bits, by writing it in binary; we need roughly $1 + \lfloor \log_2(n) \rfloor$ bits to represent a positive integer n in this way. A bit can also represent an answer to a yes-or-no question, by letting 1 mean "yes" and 0 mean"no" (or the other way around).

Quantum computers, however, have a different minimal quantity of information, based on the peculiarities of quantum mechanics. In quantum mechanics, a minimal piece of information, known as a *qubit* (short for *quantum bit*) can be a 0 or 1, but also any *superposition* of the two. We usually write our basic states 0 and 1 as *kets*,[1] which are written as $|0\rangle$ and $|1\rangle$. A superposition of our two basic states is an object of the form $a|0\rangle + b|1\rangle$, where a and b are complex numbers such that $|a|^2 + |b|^2 = 1$. We interpret this as meaning that, with probability $|a|^2$, the qubit is a $|0\rangle$, and with probability $|b|^2$, the qubit is a $|1\rangle$. The actual values of a and b, rather than just $|a|^2$ and $|b|^2$, provide some extra information, and the existence of these underlying numbers a and b is a fundamental part of the nature of quantum mechanics.

These probabilities are endemic to the nature of quantum mechanics and quantum computers, and not merely a lack of complete knowledge on our part: a qubit of the form $\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$ is not either in state $|0\rangle$ or in state $|1\rangle$, but is honestly somewhere in between. However, at any time, we can choose to measure the qubit and see whether it is a $|0\rangle$ or $|1\rangle$. When we measure a qubit of the form $a|0\rangle + b|1\rangle$, it will come up $|0\rangle$ with probability $|a|^2$ and $|1\rangle$ with probability $|b|^2$. However, once we have measured it, the superposition is lost forever: if the measurement of the qubit comes up as $|0\rangle$, say, then that means that the qubit has now changed form: it is now $1|0\rangle + 0|1\rangle$. And if we measure it more times, we will always get the same result: given a superposition v of $|0\rangle$ and $|1\rangle$, if we measure it once and get a $|0\rangle$, say, then if we measure it again we will still get $|0\rangle$. It will not be analogous to an independent coin flip. Imagine instead that the coin has glue on it, so that as soon as it hits the table, it's stuck there for good. (But, as we shall see soon, this is not quite right, because we can perform further operations on a qubit after measuring it that can alter it.)

However, we can also measure a qubit differently: let us write

$$|+\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle, \qquad |-\rangle = \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle.$$

We can express $|0\rangle$ and $|1\rangle$ in terms of $|+\rangle$ and $|-\rangle$: we have

$$|0\rangle = \frac{1}{\sqrt{2}}|+\rangle + \frac{1}{\sqrt{2}}|-\rangle, \qquad |1\rangle = \frac{1}{\sqrt{2}}|+\rangle - \frac{1}{\sqrt{2}}|-\rangle.$$

Notice that these are also valid probabilities, in that the sums of squares of the absolute values of the coefficients for $|0\rangle$ and $|1\rangle$ in terms of $|+\rangle$ and $|-\rangle$ are 1.

In fact, given *any* superposition v of $|0\rangle$ and $|1\rangle$, we can uniquely express v in terms of $|+\rangle$ and $|-\rangle$, again ending up with sensible probabilities. We can then *measure* v in terms of $|+\rangle$ and $|-\rangle$, which gives a result of either $|+\rangle$ or $|-\rangle$. When we perform this measurement, then the qubit collapses to one of the two states $|+\rangle$ and $|-\rangle$, and it no longer retains any other information about the superposition, in

[1]The word *ket* is the last part of the word *bracket*. In quantum mechanics, it is customary to use complete brackets, written $\langle a|b\rangle$, as well as both half-brackets. We call $\langle a|$ a *bra* and $|b\rangle$ a *ket*. But we only need the kets for our purposes.

terms of the probabilities of $|+\rangle$ and $|-\rangle$. However, the measured qubit retains some superposition of $|0\rangle$ and $|1\rangle$, because it is either $|+\rangle$ or $|-\rangle$: if we measure it again in terms of $|0\rangle$ and $|1\rangle$, then we will get $|0\rangle$ half the time and $|1\rangle$ half the time. Note that this is true regardless of what superposition v was originally.

So, let us start with some superposition v of $|0\rangle$ and $|1\rangle$ (or $|+\rangle$ and $|-\rangle$). Suppose we measure v in terms of $|0\rangle$ and $|1\rangle$ and get $|1\rangle$. Then, we measure the resulting qubit (which is now $|1\rangle$) in terms of $|+\rangle$ and $|-\rangle$ and get $|-\rangle$. Then, we measure the resulting qubit. We might now get $|0\rangle$. So, if we measure it twice in a row in one basis, we will always get the same result, but if we separate those two measurements by a measurement in the other basis, we could get a different result. The reason is that measurement is fundamentally a disturbance to qubits: it is not possible to measure a qubit without potentially modifying it.

All this superposition stuff may seem like an annoyance, and indeed things are generally much more complicated when working with qubits than with classical bits. But, if we are very clever, we can do new tricks with qubits that are not possible with classical bits.

18.2 Modifying Quantum States

There are some basic operations that we can perform with a classical computer. For instance, we may take a bit and turn it into a 0, regardless of what we started with. Or we may negate it: if it's a 0 then turn it into a 1 and vice versa. Or, starting with two bits, we may initialize a new bit to be their sum or product, taken modulo 2. These are the sorts of basic operations we are allowed to do with classical computers. We call these operations *logic gates*. Any more complicated arithmetic operation can be broken down into a sequence of these elementary logic gate operations.

There are also quantum versions of logic gates, which we call *quantum gates*. These are more complicated, because we can start with a qubit in some pure state and modify it so that it becomes in a superposition. The only rule when we do that is that it has to convert to a "good" basis. In linear algebra, we say that all quantum operations must be *unitary transformations*, which are generalizations of rotations, but I do not wish to get into precisely what that means here. Instead, it informally means that it behaves well with respect to superpositions. Let us see the sorts of bad things that happen with non-unitary operators:

Example Let us suppose that we were to have a quantum gate that sends $|0\rangle$ to $|\oplus\rangle = \frac{3}{5}|0\rangle + \frac{4}{5}|1\rangle$ and $|1\rangle$ to $|\ominus\rangle = \frac{24}{25}|0\rangle + \frac{7}{25}|1\rangle$. This looks fine at first: both of these are suitable as probabilities since $\left(\frac{3}{5}\right)^2 + \left(\frac{4}{5}\right)^2 = 1$, and similarly $\left(\frac{24}{25}\right)^2 + \left(\frac{7}{25}\right)^2 = 1$. But the problem is that this transformation behaves badly with respect to superpositions. What does it do to $\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$? It has to send it to $\frac{1}{\sqrt{2}}|\oplus\rangle + \frac{1}{\sqrt{2}}|\ominus\rangle$, which is

$$\frac{1}{\sqrt{2}}\left(\frac{3}{5}|0\rangle + \frac{4}{5}|1\rangle\right) + \frac{1}{\sqrt{2}}\left(\frac{24}{25}|0\rangle + \frac{7}{25}|1\rangle\right) = \frac{1}{\sqrt{2}}\left(\frac{39}{25}|0\rangle + \frac{27}{25}|1\rangle\right).$$

But this is invalid, because

$$\left(\frac{1}{\sqrt{2}}\frac{39}{25}\right)^2 + \left(\frac{1}{\sqrt{2}}\frac{27}{25}\right)^2 = \frac{9}{5} \neq 1.$$

One of the things we can do with a qubit is to change the basis, as long as the change-of-basis is one of these allowable unitary transformations. For instance, we can apply the transformation that sends $|0\rangle$ to $|+\rangle$ and $|1\rangle$ to $|-\rangle$. This is one of the most important quantum operations, and it is called the *Hadamard transform*. Any implementation of it is called a *Hadamard gate*. By direct calculation, we can see that if we apply a Hadamard gate twice to $|0\rangle$, then we end up with $|1\rangle$, whereas if we apply a Hadamard gate twice to $|1\rangle$, then we end up with $-|0\rangle$, which is the same as $|0\rangle$ after measurement. In other words, two Hadamard gates will swap $|0\rangle$ and $|1\rangle$.

When we talked earlier about *measuring* a qubit in the $|+\rangle/|-\rangle$ basis, we really mean that we apply a Hadamard gate first, and then measure it. If we apply a Hadamard gate to a qubit that is known to be either $|+\rangle$ or $|-\rangle$ and then measure and end up with $|1\rangle$, that means that before applying the Hadamard gate we had a $|+\rangle$. Similarly, if we get a $|0\rangle$ in this way, then we previously had a $|-\rangle$.

18.3 A Quantum Cryptographic Protocol

There are several ways we can get a secure quantum cryptosystem out of qubits. The first one proposed is the *BB84 protocol*, named after the proposers Charles Bennett and Gilles Brassard, and the year their paper [BB84] was written.

The idea is to construct a one-time pad with the help of the bases $|0\rangle$ and $|1\rangle$, and $|+\rangle$ and $|-\rangle$. In order to create a key, Alice creates two long random strings of (classical) bits, each of the same length N. We call these strings $a_1, \ldots, a_N$ and $b_1, \ldots, b_N$; each a_i and b_i is either 0 or 1.

Based on these two strings, Alice sets up N qubits. If $a_i = 0$, then the ith qubit q_i will either be $|0\rangle$ or $|1\rangle$, whereas if $a_i = 1$, then q_i will either be $|+\rangle$ or $|-\rangle$. If $b_i = 0$, then q_i will either be $|0\rangle$ or $|+\rangle$, whereas if $b_i = 1$, then q_i will either be $|1\rangle$ or $|-\rangle$. Thus a_i and b_i together determine whether the ith qubit will be $|0\rangle$, $|1\rangle$, $|+\rangle$, or $|-\rangle$.

After setting up all these qubits, she sends them all to Bob. Bob does not know which qubits are in the $|0\rangle/|1\rangle$ basis, and which ones are in the $|+\rangle/|-\rangle$ basis. For each qubit q_i he receives, Bob decides at random whether to measure it in the $|0\rangle/|1\rangle$ basis or in the $|+\rangle/|-\rangle$ basis. After making these measurements, the qubits are all in the bases he selected. But he does not yet know which bases they were originally in when Alice set them up.

At this point, Alice reveals the string $a_1, \ldots, a_N$ to Bob, so that he knows which qubits were set up in the $|0\rangle/|1\rangle$ basis, and which ones were set up in the $|+\rangle/|-\rangle$ basis. Revealing the a_i string can be done over a public channel. Roughly half the time, Alice and Bob will have chosen the same basis, and the other half or so of the time, they will have chosen opposite bases. So, they throw away the ones where they chose different bases. The ones where they chose the same bases then become the key: they make a binary string for their key by letting $|0\rangle$ and $|+\rangle$ be 0, and $|1\rangle$ and $|-\rangle$ be 1.

Example Let us suppose that Alice sends the following string of qubits:

$$|0\rangle\ |+\rangle\ |-\rangle\ |0\rangle\ |1\rangle\ |1\rangle\ |-\rangle\ |+\rangle\ |+\rangle\ |+\rangle.$$

Bob chooses to measure the even ones in the $|0\rangle/|1\rangle$ basis and the odd ones in the $|+\rangle/|-\rangle$ basis. So he gets

$$|-\rangle\ |1\rangle\ |-\rangle\ |0\rangle\ |-\rangle\ |1\rangle\ |-\rangle\ |0\rangle\ |+\rangle\ |0\rangle.$$

Note that when Alice and Bob chose the same basis, they got the same thing, and when Bob chose the opposite basis from Alice, he got a random result in that basis. They chose the same basis for qubits 3, 4, 6, 7, and 9, so those are used to form the key, which is then 10110.

Well, almost. One thing they would like to verify is that Eve has not intercepted the qubits. In order to do that, they perform some spot checks on the qubits for which they both chose the same basis. For example, they could spot check qubit 6, and they both report that they got $|1\rangle$, which is good!

What good does this do? Let us suppose that Eve had previously intercepted the qubits and measured them, each in one of the two bases. If she had measured qubit 6 in the $|0\rangle/|1\rangle$ basis, then she would get $|1\rangle$, just like Alice and Bob did, and there would be no way of detecting her measurement. But if she had measured it in the $|+\rangle/|-\rangle$ basis, then that qubit would *be* either $|+\rangle$ or $|-\rangle$ by the time it got to Bob, so when he measures it in the $|0\rangle/|1\rangle$ basis, there is only a 50% chance that it will be a $|1\rangle$; the other half of the time it will be a $|0\rangle$. Thus, if Eve has measured the qubits, then for each qubit in which Alice and Bob chose to measure it in the same basis, there is only a 75% chance that they agree. By spot-checking enough of the qubits at random, they will be able to detect the intrusion, except with very small probability. If there is a problem with the spot check, then they know that there has been a breach in security, and they start again with the key generation. If the spot check passes, then they remove the spot-checked bits from the key, still retaining a long enough key to use as a one-time pad in encrypting their messages.

The spot check was able to detect an eavesdropper who measures the qubits *before* Bob has a chance to do so. But what if Eve measures them after Bob does, once the correct basis has been revealed to her? She can't use the original qubits, because by the time Bob gets his hands on them, he isn't likely to give them up so that Eve can see what they are. But what if she first intercepts the qubits, makes a copy of them, and then measures the copies after she knows the correct bases?

No good! A celebrated theorem in quantum mechanics is the *no-cloning theorem* of Wootters and Zurek [WZ82] and Dieks [Die82], which says that it is impossible to create an identical copy of a qubit in an unknown state. Surprisingly, though, it *is* possible to perform an approximate cloning that gets the right result $\frac{5}{6}$ of the time; see [BH96]. This is a concern for quantum cryptography, and it forces Alice and Bob to do quite a large spot check in order to be reasonably certain that their key has not been compromised.

18.4 Quantum Computers Defeat Classical Cryptography

Now that we have seen how quantum mechanics can be useful for creating shared keys and thus for encrypting messages, let us see how it can be used for the opposite task: decrypting secret messages. In particular, we shall see how quantum computers can be used to defeat RSA, and it can also be used to decrypt ElGamal in a similar way. There are other classical cryptosystems, such as lattice-based cryptosystems, that might be quantum-resistant in the sense that we do not currently know of quantum attacks against them. However, our state of knowledge about what is and is not possible with quantum computers, even at a conjectural level, seems to be quite bad: I wouldn't bet on the lattice-based cryptosystems being immune to quantum attacks!

So, let's see how to break RSA with a quantum computer! The approach here is to attack the hard problem, namely factoring, directly: given a number n, we will produce a factor of n in polynomial time, on a quantum computer. The algorithm is called Shor's algorithm [Sho99].

As we have mentioned before, in Section 16.6, if we wish to find a factor of n other than 1 and n, it suffices to find some number $x \not\equiv \pm 1 \pmod{n}$ such that $x^2 \equiv 1 \pmod{n}$: if we have such an x, then $\gcd(x-1, n)$ and $\gcd(x+1, n)$ are factors of n. So, that is what we shall do: we shall use a quantum computer to produce an x such that $x^2 \equiv 1 \pmod{n}$ but $x \not\equiv \pm 1 \pmod{n}$, with some reasonably high probability. And if we fail, we'll just try again. Within a few tries, we will find such an x.

So, our goal is now to find an x such that $x^2 \equiv 1 \pmod{n}$ but $x \not\equiv \pm 1 \pmod{n}$. Let us suppose that n is a product of two primes, say $n = pq$, where p and q are distinct odd primes. Let us pick some random number a, between 1 and $n-1$. If $\gcd(a, n) > 1$, then that's a factor of n, and we're done. So let's assume that $\gcd(a, n) = 1$.

We will be interested in the *order* of a in $(\mathbb{Z}/n\mathbb{Z})^\times$, i.e., the least r such that $a^r \equiv 1 \pmod{n}$. By Euler's Theorem on the totient function, such an r always divides $\phi(n) = (p-1)(q-1)$. The first thing we shall do in the direction of Shor's algorithm is to explain why being able to compute orders of arbitrary elements of $(\mathbb{Z}/n\mathbb{Z})^\times$ allows us to factor n.

The first thing to note is that the order of a modulo our unknown prime factors p and q determines the order of a modulo n: the order of a modulo n is the least common multiple of the orders of a modulo p and q. If r is the order of a modulo n and r is even, then $x = a^{r/2}$ satisfies $x^2 \equiv 1 \pmod{n}$. Certainly $x \not\equiv 1 \pmod{n}$, or else the order would be smaller than r. But is $x \equiv -1 \pmod{n}$? In order for this to happen, we must have $x \equiv -1 \pmod{p}$ and $x \equiv -1 \pmod{q}$. We can determine when that happens in terms of the order of a modulo p and q. In particular, we will show that, quite often, it is *not* the case that $x \equiv -1 \pmod{p}$ and $x \equiv -1 \pmod{q}$. Whenever $x \not\equiv -1 \pmod{p}$ or $x \not\equiv -1 \pmod{q}$, we know that $x \not\equiv \pm 1 \pmod{n}$ but $x^2 \equiv 1 \pmod{n}$, and then we know how to factor n.

So, our next goal is to understand when $x \equiv -1 \pmod{p}$ and $x \equiv -1 \pmod{q}$ and show that at least one of those things fails quite often. Recall that $\mathbb{F}_p^\times$ and $\mathbb{F}_q^\times$ are cyclic groups. Let g be a generator of $\mathbb{F}_p^\times$, and let h be a generator of $\mathbb{F}_q^\times$. Then, there exist integers s and t with $1 \le s \le p-2$ and $1 \le t \le q-2$ such that $a \equiv g^s \pmod{p}$ and $a \equiv h^t \pmod{q}$. The order of a in $\mathbb{F}_p^\times$ is the smallest positive integer k such that sk is a multiple of $p-1$, and the order of a in $\mathbb{F}_q^\times$ is the smallest positive integer ℓ such that $t\ell$ is a multiple of $q-1$. We can write these numbers down more explicitly:

$$k = \frac{p-1}{\gcd(p-1, s)}, \qquad \ell = \frac{q-1}{\gcd(q-1, t)}. \tag{18.1}$$

The order of a in $(\mathbb{Z}/n\mathbb{Z})^\times$ is $r = \operatorname{lcm}(k, \ell)$. Let us suppose that r is even. Is $a^{r/2} \equiv -1 \pmod{n}$? This happens if and only if both $a^{r/2} \equiv -1 \pmod{p}$ and $a^{r/2} \equiv -1 \pmod{q}$. These happen simultaneously if and only if the largest powers of 2 dividing k and ℓ are equal, i.e., if $k = 2^i k'$ and $\ell = 2^i \ell'$, where k' and ℓ' are both odd.

Now, by (18.1), with probability $\frac{1}{2}$, k is odd, and with probability $\frac{1}{2}$, ℓ is odd, and these events are independent of one another except that at least one of them must be even, so with probability $\frac{2}{3}$, one of them is odd and one is even. Let us suppose that k is odd and ℓ is even. When this happens, we have

$$a^{r/2} \equiv 1 \pmod{p}, \qquad a^{r/2} \equiv -1 \pmod{q},$$

and vice versa if k is even and ℓ is odd. Either way, $x = a^{r/2}$ is a solution to $x^2 \equiv 1 \pmod{n}$ but $x \not\equiv \pm 1 \pmod{n}$. (The exact probability, which may be greater than $\frac{2}{3}$, depends on the number of powers of 2 dividing $p-1$ and $q-1$, but we do not need to know the exact probability, only that it is fairly large.)

What this shows is that, if we can compute the order of an element of $(\mathbb{Z}/n\mathbb{Z})^\times$ of our choice, then we can factor n with probability at least $\frac{2}{3}$. If we choose a good value of a on the first trial, then we immediately factor n. If not, pick a new one, and keep doing so until one of them is successful. With at least a $\frac{2}{3}$ chance of success at each stage, it won't take long before one of them works.

So far, we have seen that if we can compute the order an arbitrary element $a \in (\mathbb{Z}/n\mathbb{Z})^\times$, then we can factor n. Our next step is to explain how to find the order of an element using a quantum computer. In fact, Shor's algorithm computes order in a more general context. Let Q be a number, typically chosen to be a power of 2 for convenience, and let $f : \mathbb{Z} \to \mathbb{Z}/Q\mathbb{Z}$ be a function which is periodic, i.e., there is some positive integer k such that $f(x) = f(x + k)$ for all x, and assume k is minimal with this property. This means that f is entirely determined by the values $f(0), f(1), \ldots, f(k-1)$. We will assume that we know how to compute f, but that we do *not* know what k is, and our goal is to find it using a quantum algorithm. We also assume that $f(0), f(1), \ldots, f(k-1)$ are all distinct. This is actually a crucial point, and the algorithm does not work if this condition is violated.

In the case of factoring (and finding the order of an element a in $(\mathbb{Z}/n\mathbb{Z})^\times$), our function f is $f(x) = a^x \pmod{n}$. This lies in $\mathbb{Z}/n\mathbb{Z}$, rather than $\mathbb{Z}/Q\mathbb{Z}$, but this easily fixed: we treat $f(x)$ as a number from 0 to $n-1$, which we then take modulo Q, where Q is the smallest power of 2 greater than n. The values $f(0), f(1), \ldots, f(k-1)$ are all distinct, because if we have $f(i) = f(j)$ for some $0 \le i < j \le k-1$, this means that $a^i \equiv a^j \pmod{n}$, so $a^{j-i} \equiv 1 \pmod{n}$, which means that the order is a factor of $j - i$. But this is smaller than k by construction.

In order to proceed with Shor's algorithm, we need to take an interlude to discuss two ingredients: the quantum Fourier transform and continued fractions. The quantum Fourier transform is something we can only do with a quantum computer; while it is related to the classical discrete Fourier transform, the quantum version cannot be simulated on a classical computer; it is for this that quantum computers are essential. Continued fractions, on the other hand, are entirely classical.

18.5 The Quantum Fourier Transform

The computation of the order of f relies on the quantum Fourier transform. The classical discrete Fourier transform is already a very powerful tool for computation. In Section 8.3, we discussed Karatsuba's algorithm for integer multiplication, a clever trick that multiplies two n-digit numbers in $O(n^{\log_2(3)})$ operations. We also mentioned, but did not describe, some faster algorithms. The best of these rely on the classical Fourier transform. For instance, the Schönhage-Strassen algorithm multiplies two n-digit numbers in $O(n \log(n) \log\log(n))$ operations, and it does so through a clever use of the Fourier transform and its curious properties.

It stands to reason that a quantum version of the Fourier transform might be useful for quantum algorithms, and indeed it is. Here is how it works. Let us take a modulus Q, which we will assume to be a power of 2, say $Q = 2^e$. (This isn't strictly necessary, but powers of 2 work well with qubits, just as they work well with bits on classical computers.) Let us also take some integer j with $0 \le j < Q$, and write j in binary, using e digits by filling it up with 0's at the left if necessary. For instance, if $Q = 32$ and $j = 5$, then we write it in binary as 00101, using five digits. We create a quantum state out of j by using five qubits, which are initially in states $|0\rangle$, $|0\rangle$, $|1\rangle$, $|0\rangle$, $|1\rangle$, or more conveniently written as $|00101\rangle$. We abbreviate this as $|j\rangle$.

The quantum Fourier transform is the quantum operation $\mathcal{F}$ defined by

$$\mathcal{F}|j\rangle = \frac{1}{\sqrt{Q}} \sum_{k=0}^{Q-1} e^{2\pi i\; jk/Q} |k\rangle.$$

If we have a collection of qubits that are already in a superposition, say as

$$|x\rangle = \sum_{\ell=0}^{Q-1} a_\ell |\ell\rangle,$$

then

$$\mathcal{F}|x\rangle = \sum_{\ell=0}^{Q-1} \frac{a_\ell}{\sqrt{Q}} \sum_{k=0}^{Q-1} e^{2\pi i\; jk/Q} |k\rangle.$$

The quantum Fourier transform can be implemented using $O(e^2)$ primitive quantum gates.

Intuitively, the quantum Fourier transform spreads out a pure quantum state over all possible quantum states, so as to allow cancellation of unwanted terms if we are sufficiently clever. And this is what we shall see in Shor's algorithm: we use the quantum Fourier transform to keep the terms we want and delete all the terms that do not matter.

18.6 Continued Fractions

One other ingredient we will need for Shor's algorithm is a way of finding good rational approximations to real numbers. There is a classical (i.e., non-quantum) algorithm for this, based on continued fractions. A *continued fraction* is a number of the form

$$a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cfrac{1}{a_4 + \ddots}}}},$$

where the a_i's are integers, $a_i \geq 1$ for $i \geq 1$, and the expression can be either finite or infinite. Rational numbers have finite continued fraction expansions, and irrational numbers have infinite continued fraction expansions. We abbreviate this continued fraction as $[a_0; a_1, a_2, a_3, a_4, \ldots]$. Every real number has a continued fraction expression, and they are almost unique; the only failure of uniqueness is that a rational number $[a_0; a_1, \ldots, a_n]$ with $a_n > 1$ can also be written as $[a_0; a_1, \ldots, a_n - 1, 1]$.

Table 18.1 Computing the continued fraction expansion of $\frac{702}{107}$

i	0	1	2	3	4	5	6	7	8
α_i	$\frac{702}{107}$	$\frac{107}{60}$	$\frac{60}{47}$	$\frac{47}{13}$	$\frac{13}{8}$	$\frac{8}{5}$	$\frac{5}{3}$	$\frac{3}{2}$	2
a_i	6	1	1	3	1	1	1	1	2

The continued fraction expansion of a real number is also very easy to compute. Let $\alpha_0 = \alpha$. Then, for each nonnegative integer i, we have $a_i = \lfloor \alpha_i \rfloor$ and $\alpha_{i+1} = \frac{1}{\alpha_i - a_i}$.

If we truncate the continued fraction expansion after some a_i, without moving on to the next $+\frac{1}{a_{i+1}+\cdots}$, then we get a good rational approximation to α. In fact, if $\frac{s}{t}$ is one of these convergents, then it is the best rational approximation of α out of all approximations with denominator $\leq t$.

Example Let us compute the continued fraction expansion of $\frac{702}{107}$. We create a table of α's and a's, as shown in Table 18.1. We can read the a_i's off the bottom row of the table, so that $\frac{702}{107} = [6; 1, 1, 3, 1, 1, 1, 1, 2]$. The rational approximations obtained by truncating the continued fraction expansion are

$$6, 7, \frac{13}{2}, \frac{46}{7}, \frac{59}{9}, \frac{105}{16}, \frac{164}{25}, \frac{269}{41}, \frac{702}{107}.$$

We can see that these approximations are quite good; for instance,

$$\frac{702}{107} - \frac{13}{2} \approx 0.0607,$$

which is very good for an approximation by a rational number with denominator 2. Similarly,

$$\left| \frac{702}{107} - \frac{269}{41} \right| = \frac{1}{4387} \approx 0.00023.$$

Note how small this error is compared with the denominator 41 of the approximating rational number. The most impressive approximants, relative to the size of the denominator, are the ones obtained by truncating before large a_i's.

Example The continued fraction expansion of π is notable for having several unusually large a_i's at the beginning of its expansion:

$$\pi = [3; 7, 15, 1, 292, \ldots].$$

The associated convergents are the familiar good approximations for π:

$$3, \frac{22}{7}, \frac{333}{106}, \frac{355}{113}, \frac{103993}{33102}.$$

Notice that the presence of the 292 in the continued fraction expansion of π makes $\frac{355}{113}$ such a great approximation that we need to move on to well in the five digits for the denominator in order to find another comparably good approximation.

There is a lot of fascinating mathematics related to continued fractions (some of which is relevant to cryptography), but it will take us too far off track to investigate it here. Instead, we refer the interested reader to [vdP86] or [Khi97] for general properties of continued fractions and to [LP31] for an application to factoring with a classical computer.

18.7 Back to Shor's Algorithm

With the quantum Fourier transform and continued fraction method in place, we can now go through Shor's algorithm! Recall that our new goal is to find the period of a function f. In the case of factoring n, recall that we pick some number a with $1 \leq a \leq n-1$, which we may assume to be relatively prime to n (or else we can factor it directly by computing $\gcd(a, n)$), and we consider the function $f(x) = a^x \pmod{n}$, which we pad to be modulo Q, where Q is a suitable power of 2. But Shor's order-finding algorithm works more generally.

First, we initialize a qubit

$$|y\rangle = \frac{1}{\sqrt{Q}} \sum_{k=0}^{Q-1} |k\rangle.$$

This takes up $e = \log_2(Q)$ qubits. We then use a further e qubits to express $f(k)$, so that we now have $2e$ qubits, where the first e are in a superposition of $|k\rangle$ and the last e are in a superposition of $|f(k)\rangle$. We write this as

$$|y, f(y)\rangle = \frac{1}{\sqrt{Q}} \sum_{k=0}^{Q-1} |k, f(k)\rangle.$$

Next, we apply the quantum Fourier transform to the first e qubits, the ones containing $|k\rangle$. By the definition of the quantum Fourier transform, this leads to the state

$$\begin{aligned} |\mathcal{F}y, f(y)\rangle &= \frac{1}{\sqrt{Q}} \sum_{k=0}^{Q-1} \frac{1}{\sqrt{Q}} \sum_{\ell=0}^{Q-1} e^{2\pi i\, k\ell/Q} |\ell, f(k)\rangle \\ &= \frac{1}{Q} \sum_{m=0}^{Q-1} \sum_{\ell=0}^{Q-1} |\ell, m\rangle \sum_{k: f(k)=m} e^{2\pi i\, k\ell/Q} \end{aligned}$$

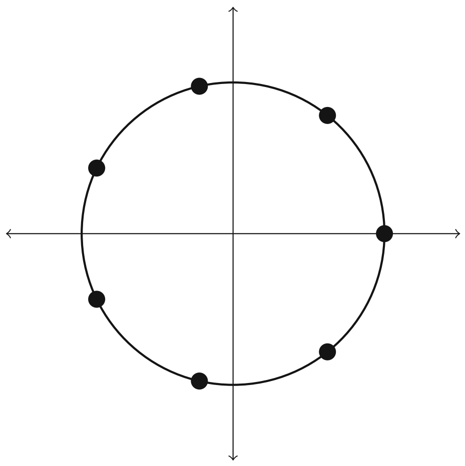


Figure 18.1 The sum of the displayed numbers $e^{2\pi i\ b/7}$ for $0 \le b \le 6$ is 0

We now perform a measurement of these $2e$ qubits. When we do this, we end up with $2e$ qubits of the form $|\ell, m\rangle$. The probability of getting this pair is

$$\left|\frac{1}{Q}\sum_{k:f(k)=m} e^{2\pi i\ k\ell/Q}\right|^2 = \frac{1}{Q^2}\left|\sum_{b=0}^{(Q-k_0-1)/r} e^{2\pi i\ \ell rb/Q}\right|^2, \qquad (18.2)$$

where k_0 is the smallest value with $f(k_0) = m$ and r is the period.

This probability is large only when $e^{2\pi i\ \ell r/Q}$ is close to 1, i.e., when $\frac{\ell r}{Q}$ is close to an integer, which we can call c. The reason for this is that, for any integer N, we have

$$\sum_{k=0}^{N-1} e^{2\pi ik/N} = 0.$$

(See Figure 18.1.) The right side of (18.2) contains a bunch of terms that cancel in this way as long as the terms go around the entire circle. The only way we can get a large sum, without much cancelation, is if we don't go all the way around the circle, which can only happen if the $b = 1$ term is very close to the positive real axis, i.e., when $e^{2\pi i\ \ell r/Q}$ is close to 1.

It follows that, after performing the measurement, we usually end up with something where $\frac{\ell r}{Q}$ is close to being an integer c. Since ℓ and Q are known, and r and c are unknown, it makes sense to rearrange a bit: $\frac{\ell}{Q}$ is close to $\frac{c}{r}$. Recall also that $r < Q$.

Now let's finish up Shor's algorithm. Given a real number α and a bound Q, the continued fraction algorithm finds *all* rational numbers $\frac{s}{t}$ such that $t < Q$ and

$$\left|\alpha - \frac{s}{t}\right| < \frac{1}{tQ}.$$

The list is pretty small, so we can test out all the rational numbers $\frac{s}{t}$ on our list and see whether any of the values of t are the period r. We can do this classically, by picking some x and just *checking* if $f(x+t) = f(x)$. If so, we're done, and we've just factored n. Otherwise, just try the whole process over with a different value of a. The "otherwise" part only happens rarely, so most of the time we will actually succeed in factoring n. And that's Shor's algorithm in a nutshell!

18.8 Problems

(1) Consider the qubit $v = \frac{3}{5}|0\rangle - \frac{4}{5}|1\rangle$. Express v in terms of the $|+\rangle/|-\rangle$ basis.

(2) Start with the qubit $v = \frac{3}{5}|0\rangle - \frac{4}{5}|1\rangle$. First measure it in the $|0\rangle/|1\rangle$ basis, and then measure the resulting state in the $|+\rangle/|-\rangle$ basis. What is the probability of getting $|+\rangle$, and what is the probability of getting $|-\rangle$?

(3) Suppose Alice sends Bob the following sequence of qubits using the BB84 protocol:

$$|+\rangle \ |0\rangle \ |+\rangle \ |-\rangle \ |-\rangle \ |1\rangle \ |+\rangle \ |0\rangle \ |+\rangle \ |1\rangle \ |1\rangle \ |1\rangle.$$

Bob measures the ones in prime position in the $|+\rangle/|-\rangle$ basis and the others in the $|0\rangle/|1\rangle$ basis. What is their shared secret key? (The numbering starts from 1, so that the first qubit is $|+\rangle$, then the second one is $|0\rangle$, and so forth.)

(4) Compute the first several terms of the continued fraction expansion for $\sqrt{2}$ and e. What is the pattern? Can you prove it? Give good rational approximations for $\sqrt{2}$ and e.

(5) Suppose we have a function $f : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}/Q\mathbb{Z}$ which is periodic, in that there exist positive integers r and s so that $f(x+r, y+s) = f(x, y)$ for all $x, y \in \mathbb{Z}$. Explain how to modify Shor's Algorithm to find r and s. Feel free to make any non-stupid assumptions on f that you want.

(6) Explain how to interpret the discrete logarithm problem in terms of order-finding. (Hint: Va beqre gb svaq n fb gung k vf pbatehrag gb t gb gur n zbqhyb c, pbafvqre gur shapgvba gung fraqf v naq w gb t gb gur v gvzrf k gb gur w zbqhyb c.) Explain how to solve the discrete logarithm problem on a quantum computer.

Chapter 19
Markov Chains

19.1 Introduction to Probability

We will now move away from public-key cryptography in the remaining chapters. Most of the mathematics used so far has been number theory and abstract algebra. But now we will see that probability is also useful in cryptanalysis. In particular, we will see a method for breaking substitution ciphers that is fully automated. It won't necessarily solve the cipher fully, but it will tend to get sufficiently close that fixing it at the end by hand will be very easy.

The first things we study in probability are usually *independent events*. Suppose we flip a coin five times. What is the probability the coin lands heads every time? We assume that the coin is fair, so that it lands on heads and tails each with probability $\frac{1}{2}$.

In order to solve this problem, we need a bit of theory, which you are likely familiar with but with less formalism. Suppose that A and B are *events*, i.e., things that can happen. (The term "event" has a technical meaning which requires a substantial amount of background to describe, but we will not need the precise version here.) We say that A and B are *independent* if learning that A has occurred does not tell you anything extra about whether B has occurred or not, and vice versa. In this case, two coin flips are independent: if the coin lands heads on the first flip, that makes it neither more nor less likely that it will land heads on the second flip.

We write $\mathbb{P}(A)$ for the probability that A occurs. We also use notations like $A \cap B$ and $A \cup B$ for the events that both A and B occur, and the probability that at least one of A and B occurs, respectively. A famous relation between these probabilities is

$$\mathbb{P}(A \cup B) = \mathbb{P}(A) + \mathbb{P}(B) - \mathbb{P}(A \cap B),$$

the simplest nontrivial case of the *inclusion–exclusion principle*. We also use the notation $\overline{A}$ for the *complement* of A: $\overline{A}$ occurs if and only if A does not. In the case that A is the event of the coin landing heads on the first flip, $\overline{A}$ would be the event of the coin not landing heads on the first flip. (This is close to the same as landing tails, but occasionally coins land on the edge.)

Proposition 19.1 *If A and B are independent, then*

$$\mathbb{P}(A \cap B) = \mathbb{P}(A) \times \mathbb{P}(B).$$

Using this Proposition, we can now solve the five-flip problem: since the flips are independent, the probability that it lands on heads five times is the product of the probabilities that it lands heads on each occasion; that is, $\left(\frac{1}{2}\right)^5 = \frac{1}{32}$.

However, not all events are independent.

Example Suppose we take a shuffled deck of cards. Let A be the event that the top card is a diamond and B the event that the second card is a diamond. Let us work out the probabilities of A, B, and $A \cap B$. There are 13 diamonds among 52 cards, so the probability that the top card is a diamond is

$$\mathbb{P}(A) = \frac{13}{52} = \frac{1}{4}.$$

By exactly the same argument, $\mathbb{P}(B) = \frac{1}{4}$. But what about $\mathbb{P}(A \cap B)$? In order for both A and B to occur, we need to get a diamond on top, which happens with probability $\frac{1}{4}$. But then for the second card, there are only 12 diamonds left, out of 51 cards, so *given that the top card is a diamond*, the probability that the second card is also a diamond is $\frac{12}{51}$. To compute the total probability, we multiply these numbers together to get

$$\mathbb{P}(A \cap B) = \frac{1}{4} \times \frac{12}{51} = \frac{1}{17}.$$

This is not the same as $\mathbb{P}(A) \times \mathbb{P}(B) = \frac{1}{16}$, so A and B are *not* independent.

The example hints at a general method for computing joint probabilities. Given events A and B, let $\mathbb{P}(A \mid B)$ be the probability that A occurs *given* that we know that B has occurred. If A and B are independent, then $\mathbb{P}(A \mid B) = \mathbb{P}(A)$, since knowing about B gives us no further insight into whether A is likely to happen. The analysis in the example holds more generally, and we have

$$\mathbb{P}(A \cap B) = \mathbb{P}(B) \times \mathbb{P}(A \mid B) = \mathbb{P}(A) \times \mathbb{P}(B \mid A).$$

(The last equality holds because A and B are symmetric.)

It is important to note that $\mathbb{P}(A \mid B)$ and $\mathbb{P}(B \mid A)$ are not necessarily the same, and can, in fact, be very different. Solving for one in terms of the other gives us Bayes's Theorem, quite possibly the only theorem to have a cult following. (I won't mention any names, but if you have to join a cult, then a cult that's mostly built around improving rationality is probably a reasonable option. On the other hand, you aren't required to join a cult.)

Theorem 19.2 (Bayes)

$$\mathbb{P}(B \mid A) = \frac{\mathbb{P}(A \mid B) \times \mathbb{P}(B)}{\mathbb{P}(A)}.$$

The fact that $\mathbb{P}(A \mid B)$ and $\mathbb{P}(B \mid A)$ are different, and potentially very different, can lead to results which at first seem counterintuitive. But remember, in math we trust.

Question 19.3 *Suppose there is a disease that 1 person in 1000 has, and there is a test for it that is correct* 99% *of the time. (That is, if you have the disease, then you will test positive* 99% *of the time; if you don't have the disease, you will test negative* 99% *of the time.) Given that you test positive, what is the probability that you have the disease?*

Answer Let D be the event that you have the disease, and T be the event that you test positive. Then we want to know $\mathbb{P}(D \mid T)$. We do this by working out all the other quantities in Bayes's Theorem: $\mathbb{P}(T \mid D)$, $\mathbb{P}(D)$, and $\mathbb{P}(T)$. We know that $\mathbb{P}(T \mid D) = .99$ and $\mathbb{P}(D) = 0.001$. But what about $\mathbb{P}(T)$? There are two ways that you can test positive: you can test positive and have the disease, or you can test positive and not have the disease. We work out these two cases separately:

$$\mathbb{P}(T \cap D) = \mathbb{P}(D) \times \mathbb{P}(T \mid D) = 0.001 \times 0.99 = 0.00099,$$
$$\mathbb{P}(T \cap \overline{D}) = \mathbb{P}(\overline{D}) \times \mathbb{P}(T \mid \overline{D}) = 0.999 \times 0.01 = 0.00999.$$

To compute $\mathbb{P}(T)$, we add those two numbers up:

$$\mathbb{P}(T) = 0.00099 + 0.00999 = 0.01098.$$

Plugging everything into Bayes's Theorem, we get

$$\mathbb{P}(D \mid T) = \frac{0.99 \times 0.001}{0.01098} \approx 0.09016.$$

So, if you test positive, the probability that you have the disease is still less than 10%.

19.2 Markov Chains

Let us suppose we wish to flip a coin n times and record the results. Let $X_1, \ldots, X_n$ denote the results of the coin flip; in each case, it will be either "heads" or "tails." The values of the X_i's are independent of each other: on the $(i + 1)$st flip, the coin does not remember the results of any previous flip. The X_i's are variables that take on *random results*. We call them *random variables*.

But what happened if it had a small amount of memory? This is hard to imagine in the case of coin flips, but it is not so hard to imagine in the case of the weather. The daily weather has a certain amount of randomness to it, but if you know today's weather, you can make a reasonable guess about tomorrow's weather. For example, consider the following simplified model of how weather changes from one day to the next:

		Tomorrow		
		Sunny	Rainy	Cloudy
Today	Sunny	.6	.1	.3
	Rainy	.3	.4	.3
	Cloudy	.5	.2	.3

(Snow? What's that? I live in California. Also, it doesn't rain in California, sadly.) This chart tells us the probabilities of what the weather will be tomorrow given that we know what it is today. For example, if it is sunny today, then the probability that it will be rainy tomorrow is .1, and the probability that it will be sunny again is .6. Observe that the sum of the entries in each *row* is 1, whereas this is not the case for the columns.

Another common way to picture the chart is as a diagram:

$$\begin{tikzcd}
& \text{Cloudy} \arrow[loop above, ".3"] \arrow[dl, ".5"] \arrow[dr, bend left, ".2"] & \\
\text{Sunny} \arrow[ur, bend left, ".3"] \arrow[rr, ".1"] \arrow[loop below, ".6"] & & \text{Rainy} \arrow[ul, ".3"] \arrow[ll, bend left, ".3"] \arrow[loop below, ".4"]
\end{tikzcd}$$

Let us write X_i for the weather on day i. The weather on day $i + 1$ depends only on the weather on day i. However, it does not really depend on what happened on previous days, except insofar as the previous days influenced the weather on day i. That is, if you know the weather on day i, then you will not be able to make better predictions about the weather on day $i + 1$ by knowing about what happened on any of the days before day i. This is the defining property of a Markov chain.

This Markov chain had three *states*, namely "sunny," "rainy," and "cloudy." In general, a Markov chain has states that live in some set S. Frequently, this set is finite; for our purposes, it will be sufficient to assume that S is finite, but this is not strictly necessary.

Definition 19.4 Let $X_1, X_2, \ldots$ be random variables taking values in a set S. If, for $s_1, \ldots, s_{n+1} \in S$, we have

$$\mathbb{P}(X_{n+1} = s_{n+1} \mid X_1 = s_1, X_2 = s_2, \ldots, X_n = s_n) = \mathbb{P}(X_{n+1} = s_{n+1} \mid X_n = s_n),$$

then $X_1, X_2, \ldots$ is said to be a *Markov chain*.

Informally, we can say that in a Markov chain, the future depends on the present, but not on the past.

In order to study a Markov chain, we use a matrix of transition probabilities like the one above. Writing it in matrix notation, we have

$$\begin{pmatrix} 0.6 & 0.1 & 0.3 \\ 0.3 & 0.4 & 0.3 \\ 0.5 & 0.2 & 0.3 \end{pmatrix}.$$

This matrix tells us how the weather changes from one day to the next.

But how does it change over several days? Given that I know the weather today, can I make predictions about what it will be not tomorrow, but the day after? Or next week?

Yes! How do we compute the probability that it will be sunny in two days given that it is cloudy today? We have to look at the different cases for what the weather will be tomorrow:

$$\begin{aligned} \text{sunny tomorrow: } & 0.5 \times 0.6 = 0.3, \\ \text{rainy tomorrow: } & 0.2 \times 0.3 = 0.06, \\ \text{cloudy tomorrow: } & 0.3 \times 0.5 = 0.15. \end{aligned}$$

The first case computes the probability that it will be sunny tomorrow and sunny in two days; the second case computes the probability that it will be rainy tomorrow and sunny in two days; and the third case computes the probability that it will be cloudy tomorrow and sunny in two days. Adding these all up, we find that the probability that it will be sunny in two days is $0.3 + 0.06 + 0.15 = 0.51$. We can compute all the other *two-step transition probabilities* in the same way.

We typically write p_{ij} for the probability $\mathbb{P}(X_{n+1} = j \mid X_n = i)$, the probability that we move from state i to state j. More generally, we write $p_{ij}^{(k)}$ for the probability that we will be in state j k steps from now, given that we're currently in state i. The previous example teaches us how to compute $p_{ij}^{(2)}$: we have

$$p_{ij}^{(2)} = \sum_k p_{ik} p_{kj},$$

where the sum is taken over all possible states. This formula makes a lot of sense: to move from i to j in two steps, we have some intermediate state k, and we sum over all k.

There is a nice way of expressing the two-step transition probabilities in terms of matrix multiplication. Recall from Chapter 6 that given an $m \times n$ matrix A whose ij entry is called a_{ij}, and an $n \times p$ matrix B whose ij entry is called b_{ij}, we define their product AB to be the $m \times p$ matrix whose ij entry is

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}.$$

Note that we cannot multiply an $m \times n$ matrix by an $\ell \times p$ matrix unless $n = \ell$.

Applying this to the case of two-step transition probabilities, if A is the matrix of one-step transition probabilities, then $A^2 = A \times A$ is the matrix of two-step transition probabilities. Note that A is an $n \times n$ matrix, so we can multiply it by itself.

Since it is annoying to multiply matrices by hand, especially if we have to do so repeatedly, we ask Sage to do it for us:

```
A = matrix([[0.6,0.1,0.3],[0.3,0.4,0.3],[0.5,0.2,0.3]])
A^2
```

Sage happily tells us that

$$A^2 = \begin{pmatrix} 0.54 & 0.16 & 0.3 \\ 0.45 & 0.25 & 0.3 \\ 0.51 & 0.19 & 0.3 \end{pmatrix}.$$

Observe that again all the rows sum to 1.

19.3 Stationarity

We might wonder what happens if we look at k-step transition probabilities for large values of k. We can compute them just as before: the matrix of k-step transition probabilities is A^k. For example, we have

$$A^{10} = \begin{pmatrix} 0.5142874014 & 0.1857125986 & 0.3 \\ 0.5142814965 & 0.1857185035 & 0.3 \\ 0.5142854331 & 0.1857145669 & 0.3 \end{pmatrix}.$$

What's going on here? It looks like all the entries in the first column are becoming very close, as are all the elements in the second column. (The third column was already all the same, and nothing will ever change that.)

This doesn't *always* happen, but it does in many interesting cases. It makes some sense that this should occur: after many steps, the Markov chain mostly forgets where it started.

Definition 19.5 We say that a Markov chain is *regular* if there is some k so that $p_{ij}^{(k)} > 0$ for all i, j.

That is, in a regular Markov chain, we can get from any state to any other state in exactly k steps, for some value of $k \geq 1$. In our weather example, we can choose $k = 1$, but this is not always the case.

Theorem 19.6 *If A is the matrix of a regular Markov chain, then*

$$B = \lim_{k \to \infty} A^k$$

exists. Furthermore, the rows of B are all equal.

The vector $v = (v_1, \ldots, v_n)$ whose entries are a row of B (any one will suffice, since they are all equal) is called the *stationary distribution* of the Markov chain. It has a very elegant interpretation.

Theorem 19.7 (Ergodic theorem) *Let $X_1, X_2, \ldots$ be a regular Markov chain with stationary distribution v. Then*

$$\frac{\#\{k \leq N : X_k = i\}}{N} \to v_i$$

as $N \to \infty$.

That is, v_i measures the proportion of time that the chain spends on state i.

19.4 Letters and Markov Chains

So far, our discussion of Markov chains has had nothing to do with letters or ciphers. As promised, though, we will now see how to attack a substitution cipher in a completely automated manner using Markov chains.

A first point is that letters can be regarded as being a Markov chain, at least approximately. The idea is that if you know one letter in a body of text, then you can make a good guess about the next letter. I computed letter frequencies using a translation of *War and Peace* that I found on Project Gutenberg; I found that the letter `h` appears around 6.6% of the time among the characters that are letters (so, I removed spaces and other similar characters before doing the count).

However, suppose we know that the previous letter was `t`. Then the probability changes dramatically: *given* that `t` just appeared, the probability that `h` appears next is around 35.1%. What a huge difference a piece of information makes! You can play a game at [Gar] to see how much a bit of knowledge makes, where you have to guess the next letter (or space) of a phrase. You will find that it is usually hard to guess the first letter of a word, but *much* easier to guess subsequent letters.

Letters do not *actually* form a Markov chain, since they do not satisfy the Markov property: if I know the last *two* letters, then I can make a better guess about the next letter than I can if I only know the last letter. Still, as a simplified model, treating letters as a Markov chain works well enough.

Modeling letter patterns was actually one of Markov's original motivations in studying Markov chains: he wanted to create a probabilistic system that had a little bit of "free will," but not too much: the letters are partially constrained by those that came before, but not entirely.

We can create a matrix of digram probabilities: the matrix whose $\alpha\beta$-entry is the probability that the next letter is a β given that the current letter is an α. (I am using α and β to denote variables whose values are *English* letters; ordinarily, I would use i and j, but those are likely to be confused for the English letters `i` and `j`.) Let us call this matrix A and write $a_{\alpha\beta}$ for the $\alpha\beta$ entry.

The matrix A feels like the matrix for a Markov chain, but actually it will not be the main Markov chain we use in order to study substitution ciphers. Let us suppose that we have a substitution cipher whose characters are $s_1, s_2, \ldots, s_N$. Here, s_i denotes the ith ciphertext character. Thus, if the ciphertext begins `GBHQEEA`, then $s_4 = \mathtt{Q}$ and $s_6 = \mathtt{E}$.

Now, a substitution cipher is a *permutation* of the letters of the English alphabet, i.e., a function

$$f : \{\mathtt{A}, \mathtt{B}, \ldots, \mathtt{Z}\} \to \{\mathtt{A}, \mathtt{B}, \ldots, \mathtt{Z}\}.$$

It is the set of permutations f on which we will run a Markov chain.

Given a function f, we want a measure of how good or plausible it is as a solution to the substitution cipher. To do this, we use the matrix A. For each pair of consecutive letters s_i, s_{i+1} in the ciphertext, we apply our permutation f to them and look at how likely that digram is; that is, we look at $a_{f(s_i), f(s_{i+1})}$. Then we multiply together these numbers over all pairs of consecutive letters in the ciphertext and get a number. The larger the number, the more plausible f is. More precisely, we set

$$\mathrm{Pl}(f) = \prod_{i=1}^{N-1} a_{f(s_i), f(s_{i+1})}.$$

Next, we try to find a good f. We do this by starting with some f, perhaps the identity permutation $f(s_i) = s_i$, and trying to improve it a bit at a time. How do we improve it? One method is to make random small changes: pick two distinct letters α and β at random, and swap what f does to α and β but keeping everything else the same; that is, we define a new permutation $\widetilde{f}$ of the letters of the alphabet by setting $\widetilde{f}(\alpha) = f(\beta)$ and $\widetilde{f}(\beta) = f(\alpha)$, and $\widetilde{f}(\gamma) = f(\gamma)$ for any $\gamma \neq \alpha, \beta$.

Now, maybe $\widetilde{f}$ is better than f, and maybe it isn't. We can check by computing their plausibility functions $\mathrm{Pl}(f)$ and $\mathrm{Pl}(\widetilde{f})$. If $\widetilde{f}$ is better than f, then we should make $\widetilde{f}$ into our new f and continue on with trying to improve it.

But what if $\widetilde{f}$ is worse than f? It would be tempting to keep the old f and forget about $\widetilde{f}$, but this is not quite the right thing to do. The problem is that we might get trapped: we might find some permutation f for which every local change makes it worse, but that is still not right. So we want to be able to escape from f if this happens.

The solution is that if $\widetilde{f}$ is worse than f, then we will *sometimes* switch to $\widetilde{f}$ and sometimes stick with f. We will switch with a certain probability, and there is a general method called the *Metropolis–Hastings algorithm* that tells us exactly what that probability should be: it is the ratios of the plausibility functions: $\frac{\mathrm{Pl}(\widetilde{f})}{\mathrm{Pl}(f)}$. With this probability, we switch from f to $\widetilde{f}$, and otherwise we keep f.

Intuitively, this makes sense: if $\widetilde{f}$ is only slightly better than f, then we will always switch, so if $\widetilde{f}$ is only slightly worse than f, we should at least usually switch. But if f is much better, we should switch rarely.

It takes many random swaps in order to hit upon the correct function (several thousand, in fact), so it is not practical to perform this algorithm by hand. But it isn't very hard to program.

This Markov chain approach to substitution ciphers is described in Persi Diaconis's paper [Dia09], where he tells the story of how two Stanford students, Marc Coram and Phil Beineke, broke a code written by a prisoner. The result is impressive, since the text is partially in English and partially in Spanish and contains a lot of slang. It would have been tough to break the code by traditional means. As part of their study of this algorithm, Coram and Beineke also tested it on a section from *Hamlet*; after 2000 iterations, the original text had been recovered, whereas after 1000 it was still a big mess.

There is much more that one can say about Markov chains; they are fascinating objects, and one can devote one's entire life to studying them. (Indeed, many people do.) But as the rest of the subject is not so applicable to cryptography, we will refrain from further discussion of Markov chains in this book.

19.5 Problems

(1) (a) Suppose you meet a person who tells you "I have two children, at least one of whom is a boy." What is the probability that both children are boys?
    (b) Now suppose that the person had said "I have two children, at least one of whom is a boy born on a Tuesday." Now what is the probability that both children are boys?
    (c) Why are the answers to these two questions different?

(2) A gambler walks into a casino with \$1. Every minute, the gambler places a \$1 bet that a fair coin, when flipped, will land heads. Thus the gambler gains \$1 if

the coin comes up heads and loses \$1 if the coin comes up tails. The gambler continues this way until either ey is out of money or has won \$5. In either of those cases, the gambler leaves the casino (with either \$5 or nothing) and goes home.

(a) Show that the amount of money the gambler has after n minutes is a Markov chain. Write down the transition matrix.
(b) For each n with $0 \le n \le 5$, what is the probability that the gambler has \$$n$ after 3 minutes?
(c) What do you think happens in the long run? Can you prove it?

(3) An ant starts on vertex A of a triangle ABC. Every second, the ant walks along one of the edges adjacent to the current vertex to arrive at a new vertex, each with equal probability. (So, for example, after 1 second, the ant is either at vertex B or vertex C, each with probability $\frac{1}{2}$.)

(a) After 5 seconds, what is the probability that the ant is at vertex A?
(b) What about after n seconds?
(c) What is the stationary distribution? Can you see this just by thinking, without doing any computation?

(4) You have an urn with n balls, k of which are red and $n - k$ of which are blue. Outside the urn, you have a large collection of extra red and blue balls. Every second, you pick a ball from the urn, discard it, and replace it with a ball of the opposite color from the outside.

(a) What is the transition matrix for the Markov chain that keeps track of the number of red balls in the urn?
(b) What is its stationary distribution? Can you see this without doing any computations?

(5) What is the expected number of times you have to flip a coin before you get 3 heads in a row, for the first time? Is the answer the same for HTT? HTH?

Chapter 20
Some Coding Theory

20.1 Introduction to Coding Theory

Coding theory sounds as though it ought to be related to cryptography, since it has a name with many of the same connotations, and superficially it seems to address a similar problem. However, this time Alice and Bob face a different kind of enemy. The problem is the following:

Problem 20.1 Alice wishes to send Bob a message, which we can think of as being a string of 0's and 1's. However, the network connection between them is not completely reliable, and sometimes a 0 will turn into a 1, or vice versa. Assuming that they know that this doesn't happen too often, how can Alice send a message in such a way that Bob can still figure out what she wanted to say?

So, now we've lost Eve: Alice and Bob are no longer worried about someone listening in on their conversation. Instead, the problem they face comes from possible mistakes in the transmission.

Remark 20.2 The standard term in the literature for an unreliable network connection is a *noisy channel*.

Let us start with the simplest case: Alice wishes to send Bob a message, and she knows that, however long a message she sends, at most one bit will get flipped. How can she send her message to Bob so that he can still read it?

One possibility is to send her message three times: only one digit can be flipped, so if in any digit all three agree, it's right, and if two out of three agree, then they're right, and the remaining one was the error.

Example If Alice wants to send the message $m = 011$, she sends 011011011. If Bob receives the message 011001011, then he can break it down into three-digit chunks:

$$011\ 001\ 011.$$

The second of these chunks is different from the other two, and in particular, it differs from the others in the second digit, marked in red. So Bob knows that this fifth digit got flipped, because it doesn't agree with the other instances of the second digit of the message. He then reads off either the first or third chunk, which is the intended message.

But this is really inefficient: she has to send a message that's three times as long as the one she wanted to send. Is there a better way?

Here is one better way: send the message twice, and then add one more digit (called a *check bit*) which is the number of 1's in the actual message, modulo 2. Bob can still recover the message: if the two instances of the message agree, then they're right. If not, then one digit in one of them was flipped, and the last bit arbitrates which one is right.

Example Again, suppose Alice wants to send the message $m = 011$. The message contains an even number of 1's, so her extra check bit is a 0. So she sends 0110110. If Bob receives the message 0110100, then he breaks it up into two 3-digit chunks plus an extra check bit:

$$011\ 010\ 0.$$

The first and second block are different, so how does Bob determine which one is right? He turns to the check bit, which tells him how many 1's the message is supposed to contain: an even number. The first block has an even number of 1's, and the second block has an odd number, so the first block must be the uncorrupted one.

So, that's an improvement, since it reduces the number of digits needed from $3n$ to $2n + 1$. Is that the best we can do?

Let us try to analyze this more theoretically. Suppose Alice wants to send k-bit messages. What is the maximum number of k-bit messages she can send such that Bob can recover them after a possible corruption?

In order to solve this, let us figure out how many messages a single message could turn into after possible corruption: any of the k bits could have been corrupted, or perhaps no bit is corrupted, for a total of $k + 1$. So, in order to be able to correct errors accurately, we need there to be at most $2^k/(k + 1)$ possible messages sent originally. The situation is most interesting when $2^k/(k + 1)$ is an integer. Since the only factors of 2^k are smaller powers of 2, this happens when $k + 1$ is a power of 2 only. So, $k = 1, 3, 7, 15, \ldots$ are the interesting cases. See Figure 20.1 for a picture in the case of $k = 3$.

We are now tempted to ask the question of whether it is actually possible to pack $2^k/(k + 1)$ messages so that any message is recoverable from any possible corruption, when $k + 1$ is a power of 2. The answer is yes: we can do so. Such encodings are called Hamming codes. They are usually explained in terms of check digits or linear algebra. These are both possible ways of doing it, but by far the most fun and, to me, enlightening, way of constructing Hamming codes is by playing games!

But we'll save that for later. For now, we'll talk about a certain type of code that is particularly nice.

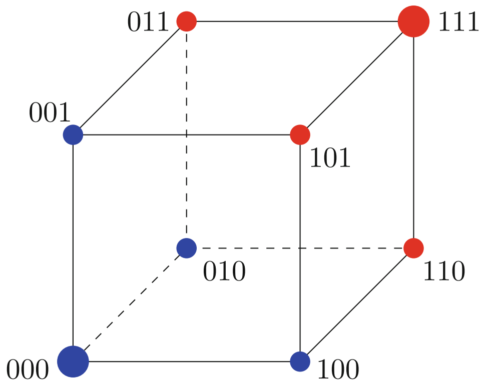


Figure 20.1 The message 000 can be corrupted into any of the four blue points, whereas the message 111 can be corrupted into any of the four red points

20.2 Linear Codes

Definition 20.3 Let F be a field, usually but not always $\mathbb{F}_2$. A *code* is a nonempty subset C of F^n, i.e., it consists of certain strings of length n, where the characters in the string are elements of F. If C is a code, then an element of C is called a *code word*.

So, if $F = \mathbb{F}_2$ and $n = 5$, then one code word might be 00000, and another might be 01001.

Definition 20.4 A code C is said to be *linear* if, whenever we have $r \in F$ and $a, b \in C$, then $a + b \in C$ and $ra \in C$.

The sum $a + b$ means to add every digit in the strings, and the product ra means to multiply every digit individually by a. So, if $F = \mathbb{F}_2$ and $a = 01001$ and $b = 11001$, then $a + b = 10000$. If $F = \mathbb{F}_p$ for some prime p, then the multiplication condition comes for free from the addition condition, so we can ignore it.

There are certain important words we use to describe nice codes:

Definition 20.5 (1) Let C be a code over a field F, and let $a \in C$ be a code word. We define the *Hamming weight* (or weight) of a to be the number of nonzero digits in a. We write $w(a)$ for the Hamming weight of a.
(2) We say that a code over $\mathbb{F}_2$ is *even* if $w(a)$ is even for all $a \in C$.
(3) We say that a code over $\mathbb{F}_2$ is *doubly even* if $w(a)$ is divisible by 4 for all $a \in C$.

One reason that we like linear codes more than other types of codes is that we can build them up from only a few code words, called generators: we start with those words, then we add them to each other, and then we add those to each other, and so forth, and all code words can be described in this way. This should remind you of what we did with elliptic curves, where we start with a few points, add them to each other to get more points, and so forth.

Remark 20.6 Every linear code contains the 0 word (the string of all 0's). To see this, suppose that a is any code word. Then $0a$ is another code word, and this word consists of all 0's.

20.3 Lexicographic Codes

One interesting type of code is called a lexicographic code. We can construct a lot of codes with interesting properties as examples of lexicographic codes, but mostly I just want to talk about them because they have such a spectacular story, and I can't bear to have people read my book without learning about them! I first learned part of the story from a video of a lecture that John Conway gave at a conference at Banff, following two papers he wrote: [CS86, Con90]. Since I enjoyed his presentation, I shall present it here in a similar spirit.

We are going to study codes in which the "digits" or "letters" are nonnegative integers. So, one of the digits might be 145; that won't count as 3 digits. However, this is just a theoretical possibility; we won't actually need to write any digits that big. Also, our strings will be infinitely long, infinite to the left, but they are finite to the right. So, an example of a string we will be considering is

$$\dots, 0, 5, 6, 10^{10^{982}} + 5, 3, 441, 0, 2.$$

However, we want all the strings we consider to start with infinitely many zeros: there are only finitely many nonzero terms at the end.

We also want to be able to put the strings in order. The ordering is lexicographic: if we have two strings, they both start with infinitely many zeros, and then at some point, they might stop being zero. So, there is a leftmost position where strings a and b differ. If, in this position, string a is less than string b, we say that $a < b$.

Example Let

$$\begin{aligned} a &: \dots, 0, 0, 1, 9, 4, 0, 1, 3 \\ b &: \dots, 0, 0, 2, 1, 0, 0, 1, 5 \end{aligned}$$

Then $a < b$, because in the leftmost position where a and b differ, string a has a 1, and string b has a 2, and $1 < 2$.

We are now ready to define the lexicographic code, or lexicode for short.

Definition 20.7 The *lexicode* consists of all strings which are lexicographically first with respect to differing from all previous strings in the lexicode in at least three positions.

Remark 20.8 There isn't anything special about 3; we could replace it with any other number larger than 1, and everything would work just the same.

From here on, we will suppress the commas because they are annoying to write, and we won't ever need any digits above 9 anyway. So, we will write something like $\cdots 01542$ to mean $\ldots, 0, 1, 5, 4, 2$.

Let's see if we can figure out some of the first few code words in the lexicode. What is the first one? It is the lexicographically first word that differs from all previous words in at least three positions. Well, there are no previous words, so anything will vacuously satisfy the differing in at least three positions condition. So, the first word in the lexicode will be the first word of all: $\cdots 00000$.

What's next? The next one has to differ from the all 0 word in at least three positions. So, we might as well take the last three positions. They can't contain any zeros, so the best we can do is to make them ones: $\cdots 00111$. Continuing on this way, we find that the first few code words are

$$\begin{array}{lll} \cdots 00000 & \cdots 00333 & \cdots 00666 \\ \cdots 00111 & \cdots 00444 & \cdots 00777 \\ \cdots 00222 & \cdots 00555 & \vdots \end{array}$$

Clearly, we can continue on with $\cdots 00nnn$, and that gives us infinitely many words, so perhaps that's the end of the story. But if it were, I probably would have skipped this story. Indeed, why do we have to stop once we have infinitely many? We can still look for the next string that differs from all of the ones we have already constructed in at least three positions. At this stage, we have to move to the fourth position from the end and put a 1 there.

Now, we can't have any two of the last three digits being the same, for if there were two n's in the last three positions, then we would be too close to $\cdots 00nnn$. So, they must all be different. The first thing to try then is $\cdots 01012$, and this is right. Continuing on looking for more words, we find that the list continues

$$\begin{array}{lll} \cdots 01012 & \cdots 01321 & \cdots 01674 \\ \cdots 01103 & \cdots 01456 & \cdots 01765 \\ \cdots 01230 & \cdots 01547 & \vdots \end{array}$$

After all of those, we can play the same game, but now we have to start with a 2. The next one is

$$\cdots 02023$$

and that will be all we need, although one could go further of course.

There is an important theorem about the lexicode:

Theorem 20.9 *The lexicode is closed under natural componentwise addition and scalar multiplication.*

This means that when we add two codewords, we get another codeword, and if we multiply every digit of a codeword by the same constant, we get another codeword.

Example Let us add the codewords $\cdots 01023$ and $\cdots 00444$.

$$\begin{array}{r} \cdots 01012 \\ + \cdots 00444 \\ \hline \cdots 01456 \end{array}$$

The resulting word, $\cdots 01456$, is another codeword. Similarly, let's multiply $\cdots 00111$ by 3.

$$\begin{array}{r} \cdots 00111 \\ \times \quad 3 \\ \hline \cdots 00333 \end{array}$$

Once again, the resulting word, $\cdots 00333$, is another codeword.

So, try to prove this theorem, or think about what some ideas might be.

This theorem has a slight problem, unfortunately, which is that it happens to be false. For example,

$$\begin{array}{r} \cdots 01012 \\ + \cdots 00111 \\ \hline \cdots 01123 \end{array}$$

which cannot be a codeword, because it's too close to $\cdots 01103$. Similarly,

$$\begin{array}{r} \cdots 01012 \\ \times 2 \\ \hline \cdots 02024 \end{array}$$

which cannot be a codeword, because it's too close to $\cdots 02023$.

In other words, *Theorem 20.9 is totally wrong*.

So, the big theorem of this section turned out to be wrong. I guess I could abandon the rest of the book or section in disgrace, but I have a better idea: I'm going to fix it. There are several ways I could fix it. One is the following:

Theorem 20.10 (Correction of Theorem 20.9) *The lexicode is* not *closed under natural componentwise addition and scalar multiplication.*

But that isn't very interesting. Besides, the original theorem was really nice, and I've decided that I want it to be true. (Ordinarily when civilized mathematicians find counterexamples to their statements, they give them up. And, as a civilized mathematician myself, I would usually do that too, but I'll sacrifice my principles here and make an exception for you, my dear reader.) So, the original theorem is going to be true.

The catch is that I tricked you in the statement: when I wrote "natural componentwise addition and scalar multiplication," you probably assumed I was going to be adding and multiplying numbers in the usual way. But that was an unwarranted assumption. Instead, I'm going to look for another way of doing addition and multiplication, and I'm going to *use* the theorem to tell me how it's done. In other words, we'll treat the theorem as an axiom and use it to build the addition and multiplication tables.

Also, let us say that we want addition and multiplication to have most of the usual properties: they should be associative and commutative, they should satisfy the distributive law, and we should have inverses with respect to addition. In short, we want to construct a ring. (That is a little weird, since the nonnegative integers aren't usually a ring, but they will be this time!)

So, let's start with the addition table. The first thing to note is that 0 must be the additive identity. To see this, note that any codeword begins with infinitely many zeros, and so the sum of any two must also. Hence $0 + 0 = 0$, so 0 is the additive identity. This allows us to start building the addition table:

+	0	1	2	3
0	0	1	2	3
1	1			
2	2			
3	3			

We can build up more of it from what we already know. For example:

$$\begin{array}{r} \cdots 01012 \\ +\cdots 00111 \\ \hline \cdots 011xy \end{array}$$

Staring at the list of code words, we find that $x = 0$ and $y = 3$. Continuing on in this way, we can find that the addition table up to 3 is:

+	0	1	2	3
0	0	1	2	3
1	1	0	3	2
2	2	3	0	1
3	3	2	1	0

We could continue it further if we wanted, but let's stop here and move onto multiplication. Since 0 is the additive identity, $0 \times x = 0$ for every x, so that helps us get started with the multiplication table:

$\times$	0	1	2	3
0	0	0	0	0
1	0			
2	0			
3	0			

Now, it isn't possible to determine the multiplicative identity from the axiom/theorem, so we have to make a choice. Fortunately, any sensible person will choose 1 to be the multiplicative identity, so we can extend our table as follows:

$\times$	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2		
3	0	3		

In order to make more progress, we look at

$$\begin{array}{r} \cdots 01012 \\ \times 2 \\ \hline \cdots 0202x \end{array}$$

Hence $2 \times 2 = 3$. To get the rest of them, remember the distributive property. So, $2 \times 3 = 2 \times (2 + 1) = 2 \times 2 + 2 \times 1 = 3 + 2 = 1$, and so on. Completing the table gives us

$\times$	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	3	1
3	0	3	1	2

And we can continue on if we like, but that's all I want to say about the integral lexicographic code. Instead, I want to tell you the tale of prisoners and their hats.

20.4 Prisoners and Their Hats

People tell many stories about the games that prison wardens force their prisoners to play involving guessing their hats, and it is possible that there are many prison wardens out there who have actually subjected their prisoners to many of these different games. But the only one I can vouch for is the following one, played in a prison far, far away.

The warden has a collection of hats, each of which is either red or blue. The warden gives each prisoner a hat at random, determined by a flip of a fair coin. Each prisoner can see everyone else's hat, but not eir own. They aren't allowed to communicate with each other. Then, each prisoner is allowed to guess eir hat color, or else pass, and all guesses must be made simultaneously and independently. If at least one prisoner guesses correctly, and no one guesses incorrectly, they are all free. Otherwise, they remain imprisoned.

The prisoners were told in advance that they were going to play this game, so they had time to work out a strategy. How can they do it?

We might expect that the best that they can do is to assign one person to guess randomly, and have all the others pass. That way, they win half the time, and they lose half the time. This feels like the optimal strategy, because each prisoner who answers has a 50% chance of being right, so we'd like as few people as possible to guess.

Remarkably, they can actually do better. Let's look at the simplest interesting case, that of three prisoners. Then, either all the prisoners have the same hat color, or else it's 2 to 1. This suggests a possible strategy: if a prisoner sees two of the same colored hats on the other prisoners, ey guesses the other color. Otherwise ey passes.

Here's what happens in all possible cases. The first three letters are the actual hat colors; the last three are the guesses. We color the wrong guesses in red and the right guesses in blue.

BBB	RRR
BBR	PPR
BRB	PRP
BRR	BPP
RBB	RPP
RBR	PBP
RRB	PPB
RRR	BBB

So they win 75% of the time, which is an improvement!

But why did our basic intuition fail us? Actually, it didn't; it just answered the wrong question. Let's look carefully: in the list above, there are six correct guesses and six incorrect guesses. The trick is that they managed to clump the wrong guesses together while spreading out the right ones. The point here is that there isn't any additional penalty for having many people guessing wrong, so there's a bit of asymmetry in the problem that they can take advantage of.

Let's figure out if we can do something like this more generally.

Suppose we can, and that there are n prisoners. Then there are 2^n possible hat configurations, and say they lose in d of these configurations in a strategy similar to the one employed above: the wrong guesses concentrated into as few losing positions as possible where everyone guesses wrong, and the right guesses spread out maximally so that in each winning position only one person guesses correctly and everyone else passes. Then they make a total of dn wrong guesses (each of n people guesses wrong in each of d sequences), and $2^n - d$ right guesses. But we need for there to be the same total number of right and wrong guesses, so we must have $dn = 2^n - d$. Solving for d, we find that $d = \frac{2^n}{n+1}$. In order for this to be an integer, $n + 1$ must be a power of 2.

Hence, the next interesting case is $n = 7$, with $d = 16$.

We can dig out a bit more from the case of $n = 3$, in fact. In each hat sequence they win, there's exactly one hat that could have been flipped that would have caused them to lose, and in all such cases, that's the person who guesses. Thus, everyone looks around and determines if it is possible that they are in a losing position. Anyone who knows that they could not will pass; anyone who is unsure will guess that they are not in a losing position.

Since each winning position is exactly one away from a losing position, every losing position must differ from every other losing position in at least three places. (Sound familiar?)

This suggests a possible idea for constructing the losing positions. For simplicity, we'll write 0 for a red hat and 1 for a blue hat, so that hat sequences are strings of zeros and ones. So, we want to write a sequence of binary strings, each of which differs from each other one in at least three spots. So, let's just try to find the "smallest" string at each stage. There's no guarantee that it will work, but it's the most obvious thing to try. Let's do this for $n = 7$. The first possible string is $S_1 = 0000000$. The first string that differs from S_1 in at least three positions is $S_2 = 0000111$. The first string that differs from S_1 and S_2 in at least three positions is $S_3 = 0011001$, and so on. Continuing on, we get the following list:

0000000	0101010	1001011	1100001
0000111	0101101	1001100	1100110
0011001	0110011	1010010	1111000
0011110	0110100	1010101	1111111

As we move on, it becomes increasingly annoying to find more strings that work by checking that each differs from each other one in at least three positions, but it can be done. In fact, I was able to write this list down quickly, in spite of not having memorized them, for reasons we will soon see.

We'll call a set C of $2^{2^n-1}/2^n = 2^{2^n-n-1}$ strings of length $2^n - 1$ with this property, that any string of length $2^n - 1$ not in C differs in one position by exactly one string in C, a *Hamming code*. These are very important in coding theory, as they are almost the only nontrivial perfect codes; the others are the *Golay codes* which we will discuss in Sections 20.9–20.11.

So, that does it for $n = 7$. Let's look at some examples.

Example Suppose the hat sequence is 0100110. Then the first person can observe that, among all listed strings, they might be in 1100110, and everyone else knows that they're not in any of them. So the first person will guess 0, and everyone else will pass, so they win.

Example If instead the hat string is 0110100, then everyone can see that they might be in the string 0110100, of all the strings above. Hence, everyone will guess that they aren't, so the guesses will be 1001011. So, everyone guesses wrong.

Now, what we would like to do is to prove that this lexicographic construction works in general, and that it produces Hamming codes. In order to do that, we must venture to the land of games, and learn about one of the two games that is worthy of serious study.

20.5 Nim

We now turn to the game of NIM. NIM is played as follows: there are two players, and there are several piles of stones on a table. Players take turns making moves, and a move consists of removing as many stones as desired (at least one), but only from one of the piles. The winner is the one who takes the last stone on the entire table.

This last sentence is interesting: sometimes people play a variant, known as MISÈRE NIM, in which the players who take the last stone *loses* the game. In the case of NIM, the strategies for these two games are extremely similar (it only requires a tiny modification to turn a winning strategy for normal NIM into a winning strategy for MISÈRE NIM), but if you change the rules a bit more (for example, you can only remove 1, 2, 3, 8, or 19 stones from a pile), then the misère version suddenly becomes very complicated to analyze, whereas the normal version is quite easy.

When we talk about NIM or some other similar game, it is useful to classify the positions as winning or losing. Except that we don't typically call them that: we classify them as winning for the next player ($\mathcal{N}$) or winning for the previous player ($\mathcal{P}$). The $\mathcal{P}$ positions are the "losing" ones.

There is a good criterion for what the $\mathcal{N}$ positions and $\mathcal{P}$ positions are:

Theorem 20.11 *An $\mathcal{N}$ position is one in which there is a move to a $\mathcal{P}$ position. A $\mathcal{P}$ position is one in which all moves are to $\mathcal{N}$ positions.*

In fact, this theorem completely characterizes the $\mathcal{N}$ and $\mathcal{P}$ positions.

Now, when you start playing NIM, you will quickly notice some tidbit of strategy: when there are two piles of the same size, they can be ignored. The reason is that the presence of these two piles allows one player to copy the other. If I have a winning strategy without those two piles, I'll just play my winning strategy for as long as you stay away from those two piles. If you remove some stones from one of those two piles, I'll just copy you in the other one, and eventually those two piles will disappear.

In short, the outcome of the game doesn't change when we add two piles of the same size, or remove two piles of the same size. Therefore, we can always assume that the number of piles with a given size is either 0 or 1.

Thus, we may represent a NIM position as a binary string: $\cdots a_5a_4a_3a_2a_1$, where a_n is the number of piles of size n, which we assume to be either 0 or 1. For example, a game with piles of size 1, 2, 3, 7 would be 1000111.

It will be helpful now for us to analyze how to play NIM in this new framework, as binary strings. There are two sorts of moves we can make in NIM: we each either remove a pile completely, or we can replace a pile with a smaller pile. The first type of move changes some 1 to a 0; the second type changes a 1 to a 0, and also changes some other number. Hence, here are the legal moves of NIM in terms of strings: we can change the string in either one or two positions. If we're ignoring repeated piles, we can also add the following restriction: the leftmost digit we change has to switch from a 1 to a 0.

Now, we have everything we need to prove the following remarkable theorem:

Theorem 20.12 *The $\mathcal{P}$ positions of* NIM *are exactly the same as the lexicographic code we constructed earlier.*

Proof There are a few things we have to check. We first check that any two $\mathcal{P}$ positions differ in at least three positions. This is pretty clear: if two $\mathcal{P}$ positions differ in at most two positions, then there's a move from one to the other, so they can't both be $\mathcal{P}$ positions.

Now, we also need to check that each $\mathcal{P}$ position is lexicographically *first* with respect to differing from each other $\mathcal{P}$ position in at most three places. To do this, we use induction: suppose we have a partial list with this property, and consider the first string which differs from all of these in at least three positions. Well, these ones are all the earlier $\mathcal{P}$ positions, and we can't get to any of them. So every move is to an $\mathcal{N}$ position, so this string must be a $\mathcal{P}$ position. ■

So, the lexicographic code, or the Hamming code, is simply the list of $\mathcal{P}$ positions in NIM!

20.6 Hamming Codes

But we're not quite done yet. We don't know that every string differs in at most one position from a $\mathcal{P}$ position. So, we must show this. Note that we haven't made any mention of the length being $2^n - 1$ yet, so that has to show up somewhere. Let us see why this is really important, by looking at the $\mathcal{P}$ positions of length 6:

000000	011001	101010	110011
000111	011110	101101	110100

There are indeed strings of length 6 which do not differ from any of these in one position, such as 001100. But, looking back to the strings of length 7, we can see that if we pad this string with an extra initial 0, then it differs from one of the *length-7* strings in one position, namely 1001100.

In order to understand this phenomenon, it is helpful to know the winning strategy for NIM. In fact, we really only need a specific feature of the winning strategy, and it is possible to prove this property without giving the full winning strategy. However, doing it that way is unappealing, since it appears to be rather unmotivated.

Anyway, here is the winning strategy for NIM. Suppose that we have piles of size $a_1, a_2, \dots, a_r$. We write each of the numbers $a_1, a_2, \dots, a_r$ in binary, and then we add without carrying. This gives us the nim sum s of the numbers.

Example Suppose we have piles of size $2, 3, 5, 7, 8$. These numbers in binary (padded with zeroes to make them all have four digits) are 0010, 0011, 0101, 0111, 1000. When we add without carrying, we get

$$\begin{array}{r} 0010 \\ 0011 \\ 0101 \\ 0111 \\ \underline{1000} \\ 1011 \end{array}$$

This is the binary representation of 11 (when written in base $10 = 9 + 1$).

The way this works is that, if $s = 0$, then the position is a $\mathcal{P}$ position; otherwise it's an $\mathcal{N}$ position. How do we make a winning move from an $\mathcal{N}$ position? We have to make it have nim sum zero after we move. The way we find such a move is to look at the leftmost 1 in the binary form of s. Then one of the original piles has a 1 in that column as well, or else that digit would be a 0. Then nim sum one of the piles with a 1 in that column with the full nim sum, and replace the full pile with one of that size.

Example In the example above, the nim sum has a 1 in the leftmost column, and the only original pile to have one there is the 8. Nim sum 8 and 11 together to get 3. So, we replace the pile of size 8 with a pile of size 3, leaving piles of size 2, 3, 3, 5, 7. This is the only winning move in that position.

Remark 20.13 More generally, if there is some n so that there is exactly one pile of size at least 2^n, then the position is guaranteed to be an $\mathcal{N}$ position. This is actually exactly what we need to prove that the lexicographic code is a Hamming code. This can be proven by induction on powers of 2 without knowing the winning strategy, but there is little to be gained from doing it that way.

What does all this have to do with Hamming codes? The point is that if we have any NIM position at all, we can add a pile of some size (possibly 0) to obtain a $\mathcal{P}$ position (the new pile we add will have size equal to the nim sum of the original piles); the only worry is how big the pile might have to be. But, by what we saw above, the new pile can't have more binary digits than the largest of the other piles.

So, that explains why the codes "close up" when their size is one less than a power of 2, but not otherwise.

20.7 Back to the Hats

We're nearly done, but let's return to the hats now. How do the prisoners figure out what they're supposed to do? Do they have to memorize the list, or is there an easy method of working out what they should guess?

What we do is to give each prisoner a number, from 1 to n, where n is the number of prisoners. Each person should write down the numbers of all the people who have blue hats and nim sum them together. There are three possibilities. Case one: we get a nim sum of 0. This will happen if the string (with that person having a red hat) is on the list, so that person should guess blue. Case two: the nim sum is eir own number. (For example, if we are prisoner number 5, and we get a nim sum of 5.) In this case, the string (with that person having a *blue* hat) is on the list, so that person should guess red. Case three: we get a nim sum that isn't 0 or our own number. In that case, pass.

In the example below, we write our hat sequences in *reverse* order, so as to correspond with our construction of the Hamming code. That is, we write $\cdots a_5 a_4 a_3 a_2 a_1$, where $a_i = 1$ is person i has a blue hat and $a_i = 0$ if person i has a red hat. For instance, when $n = 7$, the leftmost digit denotes person 7's hat color, whereas the rightmost digit denotes person 1's hat color. We also use a $\Box$ to denote an unknown hat color: person i cannot see eir own hat, so we'll put a square in position i when we refer to the sequence that person i observes.

Example Let us suppose that the actual hat sequence is 1110010, meaning that people 2, 5, 6, and 7 have blue hats. Person 1 sees 111001$\Box$ and computes the nim sum $2 \oplus 5 \oplus 6 \oplus 7 = 6$. Since 6 is not equal to 0 or 1, person 1 passes. Person 2 sees 11100$\Box$0 and computes the nim sum $5 \oplus 6 \oplus 7 = 4$ and passes, because 4 is not equal to 0 or 2. Let us now move on to person 6. Person 6 sees 1$\Box$10010 and computes $2 \oplus 5 \oplus 7 = 0$. Now, person 6 follows the rule: when the nim sum is 0, guess blue. (If the nim sum had been 6 instead, then the rule would have said to guess red.)

20.8 Linearity

The Hamming code we have constructed from NIM is a linear code. The reason for this also boils down to playing games. In order to see why, we have to define the *sum* of two games.

Definition 20.14 Let G and H be two (NIM) games. (They don't have to be NIM, but that's good enough for our purposes.) To play in the *sum* of G and H, you get to move in either G or H on your turn, but not both.

This is a reasonable definition, as NIM is already a sum of a bunch of games: it's the sum of all the individual piles. It's not always possible to determine the outcome class ($\mathcal{N}$ or $\mathcal{P}$) of the sum of games based on the outcome classes of G and H. In particular, the sum of two $\mathcal{N}$ positions might be an $\mathcal{N}$ position, and it might be a $\mathcal{P}$ position; this depends on finer structure in the game. However:

Theorem 20.15 *The sum of two $\mathcal{P}$ positions is another $\mathcal{P}$ position.*

Proof Imagine playing the sum of two $\mathcal{P}$ positions, G and H, where it's my turn to move. If I move in G, then you have a move to beat me in G, so play it. If I move in H, then you can beat me there as well, so do it. In general, just play the winning move in whichever component I play in. Eventually either G or H will disappear, and we'll be left playing the other one only. ∎

This is enough to show that the Hamming code is linear. But what about the lexicode from a while ago?

The reason that the lexicode is linear is essentially the same, but it involves playing a slightly different game, which I will call LEXNIM. How do we play LEXNIM? It is a lot like NIM played with strings, except instead of having binary strings, we have strings of nonnegative integers. Again, a move is to change the string in either one or two places, in such a way that the string decreases lexicographically. That is, the leftmost digit we change must decrease. If we change the string in two places, then the rightmost digit can change to anything, even a very large number.

Now, the same argument that we used for NIM shows that the $\mathcal{P}$ positions of LEXNIM are exactly the lexicode strings.

But linearity is now just a little bit more complicated than it was for the Hamming code, and that's because sums of LEXNIM games are slightly more complicated. We'll need a modification of a result we had earlier:

Theorem 20.16 *Let G be a game, and let H be another game which is a $\mathcal{P}$ position. Then the outcome of $G + H$ is the same as the outcome of G, i.e., if G is a $\mathcal{P}$ position, then so is $G + H$; if G is an $\mathcal{N}$ position, then so is $G + H$.*

Also, impartial games (which is the type we're considering) are their own negatives: when we add G to itself, we always get a $\mathcal{P}$ position. What we want to show now is that LEXNIM plays well with nim sums:

Theorem 20.17 *Let a and b be two positions in* LEXNIM, *and let c be the componentwise nim sum of a and b. Then the sum of the games a, b, and c is a $\mathcal{P}$ position.*

Proof We do this by showing how, after any first move, we can reduce it to another position of the same form. This proves the theorem, because by repeatedly doing this, we'll eventually get down to the position where all three strings are 0's, which is a loss.

So, how does the reduction go? If I move in a (without loss of generality, since all the strings play symmetric roles), then you play in either b or c, in the same positions that I played, in such a way to reduce the nim sum of each position to 0. In at least one of b and c, you can do so while reducing the string lexicographically. This completes the proof. ∎

So, the nim sum of two codewords is again a codeword in the lexicode. The nim sum is exactly the sum we constructed when we created this exotic addition table earlier!

As for multiplication, there's also a nim multiplication, and it has an (admittedly rather contrived-seeming) interpretation in terms of playing games, but let's leave it at that!

20.9 The Golay Codes

The Hamming code is an example of a perfect code: every string can be corrected (with at most one correction) to exactly one word in the Hamming code. One might wonder if there are others.

Let's see how to check this. Recall that, when we were looking for binary codes of length n that corrected one error, we needed that $2^n/(n+1)$ be an integer, because there are $n+1$ strings that could result from each word sent.

To generalize this, suppose we wish to be able to correct 2 errors, over the field $\mathbb{F}_p$. Then there is one way that a message can be sent correctly, there are n positions in which one error can be made, and there are $p-1$ possible errors, or $n(p-1)$ possible single errors, and $n(n-1)/2$ positions in which two errors can be made, with $(p-1)^2$ possible errors. Hence there are $n(n-1)/2 \times (p-1)^2$ ways of making two errors. Hence, there are $1 + n(p-1) + n(n-1)(p-1)^2/2$ ways of making at most two errors.

It's not easy to see when this is a factor of p^n, i.e., a power of p, but it is when $p = 3$ and $n = 11$. Similarly, if we allow three errors and work out the formula for the number of strings with that many errors, we find that we get a power of p when $p = 2$ and $n = 23$.

20.10 The Ternary Golay Code

As is the case with the Hamming codes, many people construct the Golay codes in silly ways, by listing generators of the codes and building them up from linearity. But, just like Hamming codes, the Golay codes have beautiful interpretations in terms of playing games. We just need different games.

The ternary Golay code, which corrects two errors and has length 11 over $\mathbb{F}_3$, can be interpreted in terms of a game called MATHIEU'S BLACKJACK. To play MATHIEU'S BLACKJACK, we need 12 cards, which are A through J, and a joker. The A through J are valued at 1, 2, 3, ..., 11, and the joker has value 0. To play this game, shuffle the cards, and divide them into two piles, which we'll call the main pile and the other pile. Turn them both face up.

Now, we have two players making moves. A move consists of switching one card in the main pile with one card in the other pile, but the requirement is that the card in the other pile must be lower than the card in the main pile. So, it's okay to switch an 8 in the main pile with a 2 in the other pile, but not the other way around.

The game ends when someone makes the sum of the values of the cards in the main pile drop below 21. The person who does that loses.

Our intermediate goal will be to understand the winning strategy for MATHIEU'S BLACKJACK. To do this, we need to introduce the tetracode.

The tetracode is a code over $\mathbb{F}_3$, consisting of the following strings:

0000	1012	2021
0111	1120	2102
0222	1201	2210

The rule is as follows: the first two digits determine the rest of the string: the third digit is the sum of the first two, and the fourth is the sum of the first and third, where all sums are modulo 3. In other words, the last three form an arithmetic progression, with common difference equal to the first digit.

The tetracode is an error-correcting code, in that it can correct one error. This is because the distance between any two code words is at least 3. It's also a linear code.

But the tetracode also has another error-correcting property: if we have two digits correctly placed in it, and we don't know what the other two are, then we can fill in the other two in a unique way to make it a codeword for the tetracode.

Now, to record a position in MATHIEU'S BLACKJACK, we use an object called the MiniMOG. It looks like this:

$$\begin{array}{|c|ccc|}\hline 6 & 3 & 0 & 9 \\ 5 & 2 & 7 & 10 \\ 4 & 1 & 8 & 11 \\ \hline\end{array}$$

To record a position in MATHIEU'S BLACKJACK, put a $*$ in the position of each card in the main pile. So, if we have 3, 4, 7, 8, 9, 11, then we write down the following:

$$\begin{array}{|c|ccc|}\hline & * & & * \\ & & * & \\ * & & * & * \\ \hline\end{array}$$

Now, we add each column: if there is exactly one $*$ in a column, record whether it's in the first, second, or third row. If there are exactly two $*$'s in a column, record the row that doesn't have a $*$. If there are 0 or 3 in a column, put a question mark. In this case, we obtain 2001.

We can now describe the $\mathcal{P}$ positions in MATHIEU'S BLACKJACK: they are the ones that sum to tetracode words, or ones that can be extended to tetracode words by replacing ?'s with suitable digits, and so that the number of $*$'s per column is not (3, 2, 1, 0) (in any order). Hence, a winning move is to convert a position into one that sums to a suitable tetracode.

Now, we move on to construct the ternary Golay code. Well, actually, we'll start by constructing something else first: the *extended* ternary Golay code, which is obtained from the ternary Golay code by appending a *check digit* as the final digit, so that the sum of all the digits in any code word is 0. We will start by constructing 264 words that are in the code. They will be based on the $\mathcal{P}$ positions of MATHIEU'S BLACKJACK, and we would like to put 1's in the positions of the cards in the main pile, but that doesn't quite work: because we're working over $\mathbb{F}_3$, some of them should have 1's and some should have 2's. But we're on the right track with that approach: those will be the positions of the 1's and 2's, but we have to figure out which ones should be 1's and which ones should be 2's. The rule for determining which one should be 1's and which should be 2's is pretty messy though, so I'll skip it and refer you to [LeB12, §1.10] for more information.

So, let's assume we've got that down now. Then the full extended Golay code is the linear code *generated* by these strings of weight 6, i.e., all sums of strings of this form. And, from the extended Golay code, it's easy to get to the normal Golay code: just throw away the last digit.

So, that's the ternary Golay code.

20.11 The Binary Golay Code

The other Golay code is the binary Golay code, which can correct 3 errors perfectly in a string of length 23 over $\mathbb{F}_2$. Its construction is a bit simpler than the ternary Golay code, because it is a lexicographic code. Like in the case of the ternary Golay code, there is also an extended version, called the extended binary Golay code.

The Golay code also has an interpretation in terms of a game called MOGUL. This game is played with a string of length 23, and a move consists of changing the string in up to 6 positions, in such a way that the leftmost digit of the string that is changed turns from a 1 to a 0. As is the case of NIM and the Hamming code, or with LEXNIM and the integral lexicode, the $\mathcal{P}$ positions form the binary Golay code. The extended Golay code also consists of the $\mathcal{P}$ positions of a modified version of MOGUL, played on a string of length 24 in which a move consists of changing the string in at most 7 positions.

It follows that the binary Golay code consists of all binary strings of length 23 that are lexicographically first with respect to differing from all previous strings in at least 7 positions. Similarly, the extended binary Golay code consists of all binary strings of length 24 that are lexicographically first with respect to differing from all previous strings in at least 8 positions. It is easy to convert between the two: to recover the binary Golay code from the extended binary Golay code, just throw away the last (rightmost) digit, giving us strings of length 23. To recover the extended binary Golay code from the binary Golay code, add a rightmost digit to make the Hamming weight even; that is, so that the number of 1's is even.

If all this looks to you like a lot of really cool math and you suspect that it's just a small part of an even grander story, I think you're right. Will you be the one to work out what the story is and explain it to the world?

20.12 Finite Simple Groups

One of the biggest triumphs of twentieth-century mathematics was the classification of finite simple groups. A simple group is, in a certain precise sense, a building block for all finite groups: we can take a bunch of them, glue them together, and end up with any finite group. So, in an effort to study finite groups, mathematicians initially wanted to understand what the building blocks were.

After some 15000 pages of hard mathematics spread out over hundreds of papers, mathematicians finally had their answer: the finite simple groups fall into 18 infinite families, plus 26 more so-called sporadic groups that don't fit into any family.

If you're anything like me, when you hear something like that, your reaction is going to be something like "what on earth is going on? How can we have these random objects that don't fit into any pattern? That isn't the way I expect mathematics to work!" Then you want to know the stories behind these 26 sporadic groups. And, most likely, you'll be very disappointed when you look in the literature and try to find out what the stories are, for most of them are told very badly: the explanations are things like "J_1 can be characterized abstractly as the unique simple group with abelian 2-Sylow subgroups and with an involution whose centralizer is isomorphic to the direct product of the group of order two and the alternating group A_5 of order 60." Then people will write down a bunch of matrices. However, this is not a fully satisfactory explanation for what is going on. So far, no one truly understands what these objects are trying to tell us about the world.

As a result, I haven't learned to make friends with all these 26 sporadic groups, as I would very much like to. But some of them are fairly down-to-earth objects that we can appreciate. The first five, in particular, are the Mathieu group, and they are denoted M_{11}, M_{12}, M_{22}, M_{23}, M_{24}. If the subscripts 11, 12, 23, and 24 look familiar from the Golay codes, be assured that this is no accident! The groups M_{23} and M_{24} are the *automorphism groups* of the binary Golay code and extended binary Golay code, respectively; that is, the subgroup of permutations that sends an element of the Golay code to another element of the Golay code. Similarly, M_{11} and M_{12} are related to the ternary Golay code.

There is a lot more beautiful combinatorics related to Mathieu groups and Golay codes, involving something called Steiner systems. But I shall refrain from telling it here. Instead, I will refer you to the beautiful book [CS99] for more of that story.

20.13 Problems

(1) Assuming that there is guaranteed to be at most one error in transmission, how many different messages can be sent in 5 bits? How about 6?

(2) Find a way of packing 16 messages in 7 bits, assuming there is at most one error in transmission, other than the way we did it using lexicographic codes? Can you generalize your method? That is, does it help you pack $2^{11} = 2048$ messages in 15 bits, for example?

(3) (a) Suppose that C is a linear code over $\mathbb{F}_2$, and it can be generated by code words $c_1, c_2, \ldots, c_n$. Suppose that $w(c_i)$ is even for $1 \leq i \leq n$. Show that C is an even code.
(b) If $w(c_i)$ is divisible by 4 for $1 \leq i \leq n$, is it necessarily true that C is doubly even?

(4) Suppose that C is a linear code over a field F (you can take $F = \mathbb{F}_2$ if you wish, but it shouldn't make any difference), and the minimum Hamming weight for a *nonzero* code word is a. Show that C allows you to correct $\left\lfloor \frac{a-1}{2} \right\rfloor$ errors.

(5) (a) Alice and Bob play the following game: Alice picks an integer from 1 to 100, and Bob gets to ask ten yes-or-no questions to Alice to determine the number. Alice must respond to Bob's questions, but she can tell at most one lie. Can Bob devise a strategy that guarantees that he can work out Alice's number? (His questions are allowed to be of the form "Is your number in the set $\{1, 3, 5, 19, 46, 73, 76\}$?" or similar things. In solving this question, you may assume that Alice responds in a maximally adversarial manner that is consistent with only getting to lie at most once.)
(b) What happens if Alice can pick a number from 1 to n, Bob gets to ask q questions, and Alice can tell at most ℓ lies? (This is likely too difficult in general, so find some interesting cases.)

References

[AGP94] William Robert Alford, Andrew Granville, and Carl Pomerance. 1994. There are infinitely many Carmichael numbers. *Annals of Mathematics (2)* 139 (3): 703–722. https://doi.org/10.2307/2118576.

[AH77] Kenneth Ira Appel and Wolfgang Haken. 1977. Every planar map is four colorable. I. discharging. *Illinois Journal of Mathematics* 21 (3): 429–490.

[AHK77] Kenneth Ira Appel, Wolfgang Haken, and John A. Koch. 1977. Every planar map is four colorable. II. reducibility. *Illinois Journal of Mathematics* 21 (3): 491–567.

[AHM+99] Brad Arkin, Frank Hill, Scott Marks, Matt Schmid, and Thomas John Walls. 1999. *How we learned to cheat at online poker: A study in software security*. https://www.cigital.com/papers/download/developer_gambling.php.

[Ajt96] Miklós Ajtai. 1996. Generating hard instances of lattice problems (extended abstract). In *Proceedings of the twenty-eighth annual ACM symposium on theory of computing*, 99–108. USA: ACM. https://doi.org/10.1145/237814.237838

[AKS04] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. 2004. PRIMES is in P. *Annals of Mathematics (2)* 160 (2): 781–793. https://doi.org/10.4007/annals.2004.160.781.

[BB84] Charles H. Bennett and Gilles Brassard. 1984. Quantum cryptography: public key distribution and coin tossing. In *Proceedings of IEEE international conference on computers, systems and signal processing*, 175–179.

[BH96] Vladimir Bužek and Mark Hillery. 1996. Quantum copying: Beyond the no-cloning theorem. *Physical Review A* 54 (3): 1844–1852.

[Bir68] Bryan John Birch. 1968. How the number of points of an elliptic curve over a fixed prime field varies. *Journal of the London Mathematical Society* 43: 57–60.

[Bon99] Dan Boneh. 1999. Twenty years of attacks on the RSA cryptosystem. *Notices of the American Mathematical Society* 46 (2): 203–213.

[Bre12] Otto Bretscher. 2012. *Linear algebra with applications*, 5th ed. Pearson.

[BS89] Eric Bach and Jeffrey Shallit. 1989. Factoring with cyclotomic polynomials. *Mathematics of Computation* 52 (185): 201–219. https://doi.org/10.2307/2008664.

[Cha88] David Chaum. 1988. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology* 1 (1): 65–75. http://dl.acm.org/citation.cfm?id=54235.54239.

[Cha04] David Chaum. 2004. Secret-ballot receipts: True voter-verifiable elections. *IEEE Security Privacy* 2 (1): 38–47. https://doi.org/10.1109/MSECP.2004.1264852.

[Col12] Nick Collins. 2012. World's hardest sudoku: Can you crack it? http://www.telegraph.co.uk/news/science/science-news/9359579/Worlds-hardest-sudoku-can-you-crack-it.html.

[Con] Keith Conrad. *The Miller–Rabin test*. http://www.math.uconn.edu/~kconrad/blurbs/ugradnumthy/millerrabin.pdf.

[Con90] John Horton Conway. 1990. Integral lexicographic codes. *Discrete Mathematics* 83 (2–3): 219–235. https://doi.org/10.1016/0012-365X(90)90008-6.

[Coo66] Stephen A. Cook. 1966. On the minimum computation time of functions. Ph.D. thesis. http://cr.yp.to/bib/entries.html#1966/cook.

[CS86] John Horton Conway and Neil James Alexander Sloane. 1986. Lexicographic codes: error-correcting codes from game theory. *IEEE Transactions on Information Theory* 32 (3): 337–348. https://doi.org/10.1109/TIT.1986.1057187.

[CS99] John Horton Conway and Neil James Alexander Sloane. 1999. *Sphere packings, lattices and groups*, 3rd ed., Grundlehren der Mathematischen Wissenschaften [Fundamental principles of mathematical sciences] New York: Springer. https://doi.org/10.1007/978-1-4757-6568-7. With additional contributions by Bannai, E., Borcherds, R.E., Leech, J., Norton, S.P., Odlyzko, A.M., Parker, R.A., Queen, L. and Venkov, B.B.

[Dav01] Don Davis. 2001. Defective Sign & Encrypt in S/MIME, PKCS#7, MOSS, PEM, PGP, and XML. http://static.usenix.org/publications/library/proceedings/usenix01/full_papers/davis/davis_html/.

[DF04] David S. Dummit and Richard M. Foote. 2004. *Abstract algebra*, 3rd ed. NJ: Wiley.

[DH76] Whitfield Diffie and Martin Hellman. 1976. New directions in cryptography. *IEEE Transactions on Information Theory* 22 (6): 644–654. https://doi.org/10.1109/TIT.1976.1055638.

[DHM07] Persi Diaconis, Susan Holmes, and Richard Montgomery. 2007. Dynamical bias in the coin toss. *SIAM Review* 49 (2): 211–235. https://doi.org/10.1137/S0036144504446436.

[Dia09] Persi Diaconis. 2009. The Markov chain Monte Carlo revolution. *Bulletin of the American Mathematical Society (N.S.)* 46 (2): 179–205. https://doi.org/10.1090/S0273-0979-08-01238-X.

[Die82] Dennis Geert Bernardus Johan Dieks. 1982. Communication by epr devices. *Physics Letters A* 92 (6): 271–272. https://doi.org/10.1016/0375-9601(82)90084-6. http://www.sciencedirect.com/science/article/pii/0375960182900846.

[Die11] Claus Diem. 2011. On the discrete logarithm problem in elliptic curves. *Compositio Mathematica* 147 (1): 75–104. https://doi.org/10.1112/S0010437X10005075.

[Dix81] John D. Dixon. 1981. Asymptotically fast factorization of integers. *Compositio Mathematica* 36 (153): 255–260. https://doi.org/10.2307/2007743.

[Elg85] Taher Elgamal. 1985. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory* 31 (4): 469–472. https://doi.org/10.1109/TIT.1985.1057074.

[Euc02] Euclid. 2002. *Euclid's elements*, Santa Fe, NM: Green Lion Press. All thirteen books complete in one volume, The Thomas L. Heath translation, Edited by Dana Densmore.

[Gab07] Emin Gabrielyan. 2007. The basics of line Moiré patterns and optical speedup. arXiv:physics/0703098. ArXiv Physics e-prints.

[Gar] Adriano Garsia. Shannon's experiment to calculate the entropy of English. http://www.math.ucsd.edu/~crypto/java/ENTROPY/.

[Gen10] Craig Gentry. 2010. Computing arbitrary functions of encrypted data. *Communications of the ACM* 53 (3): 97–105.

[GGH13] Sanjam Garg, Craig Gentry, and Shai Halevi. 2013. Candidate multilinear maps from ideal lattices. In *Advances in cryptology—EUROCRYPT 2013*, 7881, 1–17. Lecture Notes in Computer Science. Heidelberg: Springer. https://doi.org/10.1007/978-3-642-38348-9_1.

[GS97] William Schwenck Gilbert and Arthur Seymour Sullivan. 1997. *The complete plays of Gilbert and Sullivan*. WW Norton & Company.

[GZ86] Benedict Hyman Gross and Don Bernard Zagier. 1986. Heegner points and derivatives of L-series. *Inventiones Mathematicae* 84 (2): 225–320. https://doi.org/10.1007/BF01388809.

[Has36a] Helmut Hasse. 1936. Zur Theorie der abstrakten elliptischen Funktionenkörper I. Die Struktur der Gruppe der Divisorenklassen endlicher Ordnung. *Journal für die reine und angewandte Mathematik* 175: 55–62. https://doi.org/10.1515/crll.1936.175.55.

[Has36b] Helmut Hasse. 1936. Zur Theorie der abstrakten elliptischen Funktionenkörper II. Automorphismen und Meromorphismen. Das Additionstheorem. *Journal für die reine und angewandte Mathematik* 175: 69–88. https://doi.org/10.1515/crll.1936.175.69.

[Has36c] Helmut Hasse. 1936. Zur Theorie der abstrakten elliptischen Funktionenkörper III. Die Struktur des Meromorphismenrings. Die Riemannsche Vermutung. *Journal für die reine und angewandte Mathematik* 175: 193–208. https://doi.org/10.1515/crll.1936.175.193.

[HTZ16] B. He, A. Togbè and V. Ziegler. 2016. There is no Diophantine quintuple. *ArXiv e-prints*. arXiv:1610.04020.

[Jac04] Allyn Jackson. 2004. Comme appelé du néant-as if summoned from the void: The life of Alexandre Grothendieck II. *Notices of the American Mathematical Society* 51 (10): 1196–1212.

[Jou04] Antoine Joux. 2004. A one round protocol for tripartite Diffie-Hellman. *Journal of Cryptology* 17 (4): 263–276. https://doi.org/10.1007/s00145-004-0312-y.

[Kal03] Adam Kalai. 2003. Generating random factored numbers, easily. *Journal of Cryptology* 16 (4): 287–289. https://doi.org/10.1007/s00145-003-0051-5.

[Kar95] Anatoly Alexeevitch Karatsuba. 1995. The complexity of computations. *Proceedings of the Steklov Institute of Mathematics* 211: 169–183.

[Khi97] Khinchin, A. Ya. 1997. *Continued fractions* (Russian Edition, With a preface by B. V. Gnedenko, Reprint of the 1964 translation), xii–95, Mineola, NY: Dover Publications, Inc., ISBN: 0-486-69630-8, Mathematics Subject Classification: 11A55 (01A75 11-03 11J70), Mathematical Review Number: 1451873.

[KLM07] Phillip Kaye, Raymond Laflamme, and Michele Mosca. 2007. *An introduction to quantum computing*. Oxford: Oxford University Press.

[Kob87] Neal Koblitz. 1987. Elliptic curve cryptosystems. *Mathematics of Computation* 48 (177): 203–209. https://doi.org/10.2307/2007884.

[LeB12] Lieven LeBruyn. 2012. *Monsters and moonshine*, 114. http://win.ua.ac.be/~lebruyn/LeBruyn2012a.pdf.

[Leh59] Thomas Andrew Lehrer. 1959. Lobachevsky. Songs by Tom Lehrer, track, 6.

[Len87] Hendrik Willem Lenstra, Jr. 1987. Factoring integers with elliptic curves. *Annals of Mathematics (2)* 126 (3): 649–673. https://doi.org/10.2307/1971363.

[LHA+12] Arjen Lenstra, James P Hughes, Maxime Augier, Joppe Willem Bos, Thorsten Kleinjung, and Christophe Wachter. 2012. *Ron was wrong, whit is right*. Technical report, IACR.

[LP31] Derrick Henry Lehmer and Richard E Powers. 1931. On factoring large numbers. *Bulletin of the American Mathematical Society* 37 (10): 770–776.

[Lut37] Elisabeth Lutz. 1937. Sur l'équation $y^2 = x^3 - Ax - B$ dans les corps p-adiques. *Journal für die reine und angewandte Mathematik* 177: 238–247. https://doi.org/10.1515/crll.1937.177.238.

[Mar01] Leo Marks. 2001. *Between silk and cyanide: A codemaker's war, 1941–1945*. https://books.google.com/books?id=I4zP8hSxIFIC. Free Press.

[Maz77] Barry Mazur. 1978. Modular curves and the Eisenstein ideal. *nstitut des Hautes Études Scientifiques Publications Mathématiques* (47): 33–186. http://www.numdam.org/item?id=PMIHES_1977__47__33_0.

[Mil76] Gary L. Miller. 1976. Riemann's hypothesis and tests for primality. *Journal of Computer and System Sciences* 13 (3): 300–317. Working papers presented at the ACM-SIGACT Symposium on the Theory of Computing (Albuquerque, N.M., 1975).

[Mil86] Victor S. Miller. 1986. Use of elliptic curves in cryptography. In *Advances in cryptology—CRYPTO '85 (Santa Barbara, Calif., 1985)*, 417–426. Lecture Notes in Computer Science. Berlin: Springer. https://doi.org/10.1007/3-540-39799-X_31.

[Mor22] Louis Joel Mordell. 1922. On the rational solutions of the indeterminate equations of the third and fourth degrees. *Proceedings of the Cambridge Philosophical Society* 21: 179–192.

[Mor12] Pieter Moree. 2012. Artin's primitive root conjecture—a survey. *Integers* 12 (6): 1305–1416. https://doi.org/10.1515/integers-2012-0043.

[NC00] Michael A. Nielsen and Isaac L. Chuang. 2000. *Quantum computation and quantum information*. Cambridge: Cambridge University Press.

[Nef03] C. Andrew Neff. 2003. Verifiable mixing (shuffling) of ElGamal pairs. VHTi Technical Document. http://courses.csail.mit.edu/6.897/spring04/Neff-2004-04-21-ElGamalShuffles.pdf, VoteHere, Inc..

[NNR99] Moni Naor, Yael Naor, and Omer Reingold. 1999. Applied kid cryptography or how to convince your children you are not cheating. In *Proceedings of Eurocrypt 94*, 1–12.

[NS95] Moni Naor and Adi Shamir. 1995. chapter Visual cryptography. In *Advances in Cryptology — EUROCRYPT'94: workshop on the theory and application of cryptographic techniques perugia, Italy, May 9–12, 1994 Proceedings*, 1–12. Heidelberg: Springer. https://doi.org/10.1007/BFb0053419.

[NZM91] Ivan Niven, Herbert S. Zuckerman, and Hugh L. Montgomery. 1991. *An introduction to the theory of numbers*, 5th ed. New York: Wiley.

[Ped91] Torben Pryds Pedersen. 1991. A threshold cryptosystem without a trusted party. In *Proceedings of the 10th annual international conference on theory and application of cryptographic techniques, EUROCRYPT'91*, 522–526. Heidelberg: Springer. http://dl.acm.org/citation.cfm?id=1754868.1754929.

[PH78] Stephen Pohlig and Martin Hellman. 1978. An improved algorithm for computing logarithms over gf(p) and its cryptographic significance (corresp.). *IEEE Transactions on Information Theory* 24 (1): 106–110. https://doi.org/10.1109/TIT.1978.1055817.

[Pol74] John M. Pollard. 1974. Theorems on factorization and primality testing. *Proceedings of the Cambridge Philosophical Society* 76: 521. https://doi.org/10.1017/S0305004100049252.

[Pol75] John M. Pollard. 1975. A Monte Carlo method for factorization. *Nordisk Tidskr. Informationsbehandling (BIT)* 15 (3): 331–334.

[Pom82] Carl Pomerance. 1982. Analysis and comparison of some integer factoring algorithms. In *Computational methods in number theory, Part I of math centre tracts*, vol. 154. Amsterdam: Math. Centrum.

[Pom10] Carl Pomerance. 2010. Primality testing: Variations on a theme of Lucas. *Congressus Numerantium* 201: 301–312.

[PPVM16] Jennifer Park, Bjorn Poonen, John Voight, and Melanie Matchett. 2016. A heuristic for boundedness of ranks of elliptic curves. arXiv:1602.01431. ArXiv e-prints.

[QGAB89] Jean-Jacques Quisquater, Louis Guillou, Marie Annick, and Tom Berson. 1989. How to explain zero-knowledge protocols to your children. *Proceedings on advances in cryptology, CRYPTO '89*, 628–631. New York: Springer. http://dl.acm.org/citation.cfm?id=118209.118269.

[Rab80] Michael O. Rabin. 1980. Probabilistic algorithm for testing primality. *Journal of Number Theory* 12 (1): 128–138. https://doi.org/10.1016/0022-314X(80)90084-0.

[RSA78] Ronald Linn Rivest, Adi Shamir, and Leonard Adleman. 1978. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM* 21 (2): 120–126. https://doi.org/10.1145/359340.359342.

[Sei] Raimund Seidel. Understanding the inverse Ackermann function. http://cgi.di.uoa.gr/~ewcg06/invited/Seidel.pdf.

[Sha71] Daniel Shanks. 1971. Class number, a theory of factorization, and genera. In *1969 Number Theory Institute (Proceedings of symposia in pure mathematics, vol. XX, State University New York, Stony Brook, NY, 1969)*, 415–440. Providence: American Mathematical Society.

[Sho99] Peter W. Shor. 1999. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Review* 41 (2): 303–332. https://doi.org/10.1137/S0036144598347011.

[Sie14] Carl L. Siegel. 2014. Über einige Anwendungen diophantischer Approximationen [reprint of Abhandlungen der Preußischen Akademie der Wissenschaften. Physikalisch-mathematische Klasse 1929, Nr. 1]. In *On some applications of Diophantine approximations of Quad/Monogram*, vol. 2, 81–138. Pisa: Ed. Norm.

[Sma99] Nigel P. Smart. 1999. The discrete logarithm problem on elliptic curves of trace one. *Journal of Cryptology* 12 (3): 193–196. https://doi.org/10.1007/s001459900052.

[Sor] Sorting algorithm animations. http://www.sorting-algorithms.com/.

[SS71] Arnold Schönhage and Volker Strassen. 1971. Schnelle Multiplikation grosser Zahlen. *Computing (Archeive Elektron Rechnen)* 7: 281–292.

[Ste75] Nelson M. Stephens. 1975. Congruence properties of congruent numbers. *Bulletin of the London Mathematical Society* 7: 182–184.

[SW17] Jonathan Sorenson and Jonathan Webster. 2017. Strong pseudoprimes to twelve prime bases. *Mathematics of Computation* 86 (304): 985–1003. https://doi.org/10.1090/mcom/3134.

[Tea17] The Sage Development Team. 2017. Sage tutorial. http://doc.sagemath.org/pdf/en/tutorial/SageTutorial.pdf.

[Tun83] Jerrold B. Tunnell. 1983. A classical Diophantine problem and modular forms of weight $3/2$. *Inventiones Mathematicae* 72 (2): 323–334. https://doi.org/10.1007/BF01389327.

[vdP86] Alf J. van der Poorten. 1986. An introduction to continued fractions. In *Diophantine analysis (Kensington, 1985) of London Mathematical Society Lecture Note Series*, vol. 109, 99–138. Cambridge: Cambridge University Press.

[Wat12] Mark Watkins. 2012. Some remarks on Heegner point computations. *Explicit methods in number theory of Panor Synthèses*, vol. 36, 81–97. Paris: Mathematical Society of France.

[Wil82] Hugh C. Williams. 1982. A $p+1$ method of factoring. *Mathematics of Computation* 39 (159): 225–234. https://doi.org/10.2307/2007633.

[Wri98] Fred B. Wrixon. 1998. *Codes, ciphers & other cryptic & clandestine communication: Making and breaking secret messages from hieroglyphs to the internet*. Black Dog & Leventhal Publication.

[WZ82] William K. Wootters and Wojciech H. Zurek. 1982. A single quantum cannot be cloned. *Nature* 299 (5886): 802–803.

Index

A
Abelian groups, 85–87, 143, 146, 149, 150, 153, 154, 164, 249
Abel, Niels Henrik, 85
Agrawal–Kayal–Saxena (AKS) algorithm, 81, 137–139
Alford, Red, 136
Ali Baba, 176, 177
Appel, Kenneth, 178
Artin's primitive root conjecture, 91
Autokey cipher, 63, 65, 67, 71, 72

B
Baby-step giant-step algorithm, 101, 103–105
Bayes's Theorem, 222, 223
BB84 protocol, 210
Beineke, Phil, 229
Bézout, Étienne, 33
Bézout's Lemma, 33, 34
Big O notation, 75, 77, 83, 104, 139, 140
Birch and Swinnerton–Dyer conjecture, 165, 168
Bit, 207
Blakley's secret-Sharing scheme, 193
Bubble sort, 80

C
Cæsar cipher, 2, 5–10, 19, 41, 43, 48
Cæsar, Julius, 2, 5
Carmichael number, 92, 97, 135, 136, 140
Chaum–Pederson protocol, 202–204
Chinese Remainder Theorem, 34, 105, 106, 111, 160, 204
Chosen ciphertext attack, 115
Chosen plaintext attack, 115
Ciphertext, 6–10, 14–16, 20, 21, 41–45, 48–50, 53, 54, 63, 64, 72, 101–103, 111, 114–116, 119, 124–126, 151, 195, 201, 228
3-colorability, 178, 180, 183, 189
Columnar transposition cipher, 65, 66
Commitment scheme, 186–189
Congruent number, 165–168
Continued fraction, 215–217
Conway, John, 234
Coram, Marc, 229
Crib, 115
Cryptanalysis, 7, 30, 153
cryptanalysis, 221
Cryptosystem, 6, 7, 68, 69, 71, 92, 101, 103, 113, 115, 118, 126, 127, 150, 195

D
Determinant, 57
Diaconis, Persi, 229
Diffie–Hellman key exchange, 92, 99–103, 111, 113, 127, 150, 151, 153, 154, 195, 202
Diffie–Hellman key exchange, tripartite, 153
Diffie–Hellman Problem, 101
Digital signature, 118, 119, 125
Dining cryptographer problem, 204
Diophantine equation, 157, 165, 167
Diophantine m-tuple, 169
Dirichlet's Theorem on primes in arithmetic progression, 38
Discrete Logarithm Problem, 99–101, 103–106, 111, 139, 140, 150, 153, 183, 184, 188
Divisibility sequence, 170

Double-layer cipher, 67

E
ElGamal cryptosystem, 101–103, 111, 113, 116–118, 150, 151, 197–203
Elkies, Noam, 164
Elliptic curve cryptography, 118
ergodic theorem, 227
Euclid, 25
Euclidean algorithm, 32–35, 86, 96, 101, 102, 111, 114, 121
Euler's criterion, 182, 183
Euler's Theorem on the totient function, 90
Euler totient (ϕ) function, 37, 95

F
Fermat's Last Theorem, 89, 165
Fermat's Little Theorem, 89, 90, 97, 135, 157, 170
Fibonacci number, 170
Four-color theorem, 178
Frequency analysis, 12, 15, 19, 41, 43, 45
Friedman's index of coincidence, 49, 50, 52
Frobenius automorphism, 171
Fundamental Theorem of Arithmetic, 26, 28

G
Garg, Sanjam, 153
Generalized Riemann Hypothesis, 82, 137
Gentry, Craig, 115, 153
Gilbert and Sullivan, 19
Golay code, 246–249
Granville, Andrew, 136
Gross–Zagier formula, 165
Grothendieck, Alexander, 136

H
Hadamard gate, 210
Hadamard transform, 210
Haken, Wolfgang, 178
Halevi, Shai, 153
Hamiltonian circuit, 189
Hamming code, 232, 240, 242–246
Hasse–Weil bound, 148, 161, 163
Hill cipher, 55, 58, 60, 62
Homomorphic encryption, 115

I
Index calculus, 107, 110, 111, 153
Inverse Ackermann function, 77

J
j-invariant, 155

K
Karatsuba's algorithm, 78, 79, 83, 214
Kasiski examination, 43, 63
Kerckhoff's principle, 7
Ket, 208
Key, 7, 10, 33, 41–43, 45–47, 49–54, 63–65, 67–72, 82, 99, 101, 103, 111, 113, 114, 116–119, 121, 124–128, 130, 137, 141, 153, 154, 161, 173, 186, 192, 194–198, 200–203, 221
Koblitz, Neal, 150

L
Lagrange interpolation formula, 192, 193
Lattès map, 146
le chiffre indéchiffrable, 41
Lehrer, Tom, 48
Lenstra, Arjen, 121, 122
Lenstra, Hendrik, 26
Lenstra's elliptic curve factorization, 161–163
Lexicographic code, 234–236, 238, 242, 245, 248
Lexnim, 245
Lobachevsky, 48
Logic gate, 209

M
Markov chain, 221, 223–225, 227–230
Mathieu's blackjack, 246–248
Mazur's torsion theorem, 164
Merge sort, 80
Miller–Rabin, 81, 135–137, 139, 182
Miller, Victor, 150
Mogul, 248
Multiplicative function, 94, 95, 98

N
Nagell–Lutz theorem, 168
Nim, 241–245
NP-complete, 178, 180, 183, 189

O
One-time pad, 67, 68, 195

P
Padding, 116, 117, 125
Park, Jennifer, 165
Pell equation, 170
Permutation polynomial, 204
Plaintext, 6–9, 20, 21, 41–45, 48, 49, 51–53, 63–65, 71, 72, 101–103, 113, 115, 116, 118, 124, 125, 195, 200, 203
Pocklington–Lehmer test, 169
Pohlig–Hellman algorithm, 105, 106, 111, 112
Pollard ρ algorithm, 128–130, 139, 140
Pollard's $p - 1$ factorization, 157, 162, 171
Pomerance, Carl, 131, 136
Poonen, Bjorn, 165
Primitive root, 91, 97, 99, 102, 105, 111, 184, 188, 198
Pythagorean triple, 142

Q
Quadratic sieve, 130, 131, 133, 139
Quantum computer, 207
Quantum cryptography, 210
Quantum Fourier transform, 214, 215
Quantum gate, 209
Qubit, 208

R
Rabin cryptosystem, 126
Rail fence cipher, 65, 66

S
Safe prime, 106, 107, 159
Schnorr protocol, 183, 184
Schönhage-Strassen algorithm, 78, 214
Secret sharing, 191, 198
Shamir's secret-Sharing scheme, 192, 204
Shor's algorithm, 212, 217–219
Siegel's theorem, 168
Smooth number, 130–132, 135, 140, 163
Sophie Germain prime, 106, 107
Spivak pronoun, 70
Substitution cipher, 9–12, 19, 21, 22, 30, 41, 43, 48, 50, 52, 54, 117
substitution cipher, 221, 227–229
Sudoku, 180–182, 189
Superposition, 208
Sylvester, James Joseph, 37

T
Toom–Cook algorithm, 78, 83
Transposition cipher, 65, 66
Tunnell's theorem, 167

U
Unitary transformation, 209

V
Vigenère cipher, 41–44, 48–54, 63, 67, 103
Visual cryptography, 191, 194, 196, 200
Voight, John, 165

W
Where's Waldo, 173, 174, 176
Wood, Melanie Matchett, 165

Z
Zero-knowledge proof, 173–175, 177–180, 183, 185, 186, 188, 189, 202, 204