Cryptology

Sarang Noether, Ph.D.

Contents

Chapter 1

Introduction

Cryptology is the study of hidden messages. Its roots trace back to antiquity. There are two subfields within cryptology. *Cryptography* is the study of making secret writing, and *cryptanalysis* is the study of breaking secret writing. We will discuss both fields extensively in this course.

1.1 Terminology

We will use several terms throughout the course. Let us define them now.

Definition 1.1. *Encryption* refers to the process of making a readable message unreadable. We sometimes use the word *encoding*. Similarly, *decryption* refers to the process of making an unreadable message readable. We sometimes use the word *decoding*.

Definition 1.2. A readable message is called *plaintext*. The encrypted form of the message is called *ciphertext*.

We often write plaintext letters in lowercase, and write ciphertext letters in uppercase. In this text, plaintext is written in standard type and ciphertext is `WRITTEN IN UPPERCASE MONOSPACED` type.

Definition 1.3. A *cryptographic algorithm* is a set of steps used for encrypting or decrypting a message. We also use the terms *cryptographic system* or *cryptosystem*.

There are two basic ways to encrypt a message, codes and ciphers.

Definition 1.4. A *code* or *cipher* encrypts a message by exchanging plaintext for ciphertext according to a particular algorithm.

Definition 1.5. The *key* for a cipher is a piece of secret information needed for encryption or decryption.

The goal of the cryptographer is to encrypt a message using a scheme such that the recipient can easily decrypt the message using the known key, but no one else can. The goal of the cryptanalyst is to determine the content of a message without necessarily knowing the key.

Definition 1.6. The process of decrypting a piece of ciphertext without knowing the key is called *cracking* or *breaking* the cipher. We call such a process an *attack* on the cryptosystem.

There are two primary types of classical ciphers.

Definition 1.7. A *transposition cipher* encrypts a message by changing the position of letters in a plaintext message.

Definition 1.8. A *substitution cipher* replaces letters with different letters. In this case, the letters used in plaintext form the *plaintext alphabet* and the corresponding ciphertext letters form the *ciphertext alphabet*.

Modern ciphers combine both of these, often in very complex ways.

1.2 Exercises

1. Describe several advantages and disadvantages of codes.
2. Describe several advantages and disadvantages of ciphers.
3. What is the key for a code?

Chapter 2

Transposition ciphers

There are many types of transposition ciphers, and it is easy to invent rules for new ones.

2.1 Railfence cipher

In the *railfence cipher*, text is written (with spaces removed) alternating between two lines. The resulting lines are read out like in a book to form the ciphertext. Decryption is performed by splitting the ciphertext in half and alternating between the two sides.

Example 2.1. Suppose we want to encrypt the message "this is a railfence example" using the railfence cipher. We write the message in two alternating lines:

t i i a a l e c e a p e
h s s r i f n e x m l

Reading each line like a book, we obtain the ciphertext `TIIAALECEAPEHSSRIFNEXML`.

Example 2.2. If we use a railfence cipher to decrypt `FRYEREITOOOTDGESSOHT`, we obtain "forty degrees is too hot" as plaintext.

The railfence cipher could be extended to more than two lines. For example, in *railfence-3*, we alternate line direction up and down in a zigzag pattern covering three lines. The key to a railfence cipher, then, is the number of lines used.

2.2 Route cipher

In a *route cipher* or *path cipher*, plaintext with spaces removed is written in a grid as in a book, and read off according to a specified pattern. Any remaining blank spots are filled in with some junk character. The key corresponds to the grid size and pattern used.

Example 2.3. Suppose we write the plaintext message "once upon a midnight dreary" in a 5×5 grid. For the pattern, suppose we start at the bottom left and spiral clockwise. This gives the following key grid:

o	n	c	e	u
p	o	n	a	m
i	d	n	i	g
h	t	d	r	e
a	r	y	x	x

Applying the pattern results in the ciphertext message `AHIPONCEUMGEXXYRTDONAIRDN`.

Example 2.4. If we decrypt `CANXUOYFHMEICTAC` using a route cipher and the route starts at the bottom left and zigzags to the right and up, we obtain "catch me if you can x" as plaintext.

2.3 Exercises

1. Determine a method for decrypting railfence-3. Can you extend your method to any number of lines?
2. Crack each of the following transposition ciphers:
   (a) `TEUEBDSHDDAIE`
   (b) `CUOYARTHISKBTNEA`
   (c) `THRFNESMSEEISLECISIPLD`
   (d) `NGEIVVEERYGOOUNUNPA`

Chapter 3

Caesar cipher

The simplest substitution cipher is the *Caesar cipher* or *Caesar shift*. This cipher was reportedly (but not proven to be!) used by Julius Caesar to hide his military communications from his enemies. Though it is simple and easy to break, the fact that many in his time were illiterate made the cipher more secure.

In his original implementation, Caesar shifted each letter in his plaintext message three letters forward in the alphabet, wrapping around if he reached the end. Hence, we would have the mappings a $\mapsto$ `D` and y $\mapsto$ `B`. Thus, the plaintext message "attack at dawn" becomes `DWWDFN DW GDZQ` in ciphertext under the Caesar shift. To decrypt, shift each letter back three places in the alphabet, wrapping around if we reach the beginning.

Example 3.1. The ciphertext `SHUKDSV WKLV LV WRR HDVB` decrypts to "perhaps this is too easy" in plaintext under the classic Caesar shift.

Why use only a shift of three letters? We could easily choose any number of letters to shift, making sure we wrap around the alphabet correctly. In this case, the key to the general Caesar shift is the number of letters shifted. We might indicate, for example, the classic Caesar shift by the mapping a $\mapsto$ D or by its numeric shift of three letters.

Example 3.2. The plaintext "how much wood could a woodchuck chuck" encrypts to the ciphertext `OVD TBJO DVVK JVBSK H DVVKJOBJR JOBJR` if we map a $\mapsto$ `H`. This is a numeric shift of seven letters.

A useful tool for performing Caesar shifts is the Caesar wheel. It contains two rotating wheels, one placed over the other and fastened together. The wheel is rotated according to the chosen key so that encryption and decryption can be performed by referencing plaintext letters on one wheel and the corresponding ciphertext letters on the other wheel. Appendix A contains a Caesar wheel.

Example 3.3. Under a Caesar shift that maps a $\mapsto$ `V`, the ciphertext `BJO DO` decrypts to "got it" in plaintext.

Example 3.4. Here are the corresponding letters for several numeric shifts:

- 12: a $\mapsto$ `M`
- -4: a $\mapsto$ `W`
- 0: a $\mapsto$ `A`
- 18: a $\mapsto$ `S`

A common technique to break a general Caesar shift is called the *stream method*. In this method, successive single shifts are performed on the first few letters of ciphertext, writing them below the ciphertext. This is easy to do by continuing along in the alphabet column by column. The cryptanalyst hopes that one of the shifts gives recognizable plaintext, and continues to decrypt using that shift.

Example 3.5. Suppose we are given Caesar ciphertext `CLROPZLOBXKAPBSBKVBXOPXDL` and wish to crack it. The first few streams give the following potential plaintexts:

```
CLROPZLOBXKAPBSBKVBXOPXDL
DMSPQA...
ENTQRB...
FOURSC...
```

It looks like a shift of four letters gives recognizable words! We decrypt to get "four score and seven years ago" in plaintext. Since we shifted four letters forward (or equivalently, 22 letters backward) to get the plaintext, the original shift was 22 letters, or the mapping a $\mapsto$ `X`.

3.1 Exercises

1. Give three numerical equivalents to each of the following shifts, with at least one negative:
   (a) a $\mapsto$ `H`
   (b) q $\mapsto$ `G`
   (c) z $\mapsto$ `A`
   (d) a $\mapsto$ `Z`
   (e) j $\mapsto$ `J`
2. Decrypt the following Caesar cipher, where the plaintext was encrypted with the shift a $\mapsto$ `F`:
   ```
   BJQHTRJ YT DTZW KNWXY HTIJ. YMJWJ FWJ RFSD RTWJ YT HTRJ. FWJ DTZ WJFID?
   ```
3. Crack the following Caesar cipher:
   ```
   MHY VBA PU AOL BUJOHYALK IHJRDHALYZ VM AOL BUMHZOPVUHISL LUK VM AOL DLZALYU ZWPYHS
   HYT VM AOL NHSHEF SPLZ H ZTHSS, BUYLNHYKLK FLSSVD ZBU.
   ```
4. Use the stream method to crack the following Caesar cipher:
   ```
   LWNBPZTIGXAAXDCHLWTCLTRPCBPZTQXAAXDCH
   ```
5. Crack the following Caesar cipher:
   ```
   FIOXFS MBION NBY GUACW QILXM UHX QCH U ZUVOFIOM JLCTY. NBY GUACW QILXM ULY JOLJFY
   WIQ. VON SIO BUPY NI MBION NBYG!
   ```
6. Crack the following Caesar cipher:
   ```
   CPF VJG RQQT NKVVNG QQORC NQQORCU YGTG UQ UOCNN CPF JGNRNGUU, VJGA YQWNF IGV IQDDNGF
   WR TKIJV CPF NGHV. C YCPIFQQFNG YQWNF GCV VGP QH VJGO HQT DTGCMHCUV CPF VJKPM
   PQVJKPI QH KV. CPF UQ, K UCKF, "EQOG CPF NKXG YKVJ OG KP RGCEG CPF UCHGVA, CYCA HTQO
   CNN VJG YCPIFQQFNGU, CPF JQTPUYQIINGTU, CPF UPQBBYCPIGTU, CPF TQVVGP, XGTOKEKQWU
   MPKFU."
   ```
7. Crack the following Caesar cipher:
   ```
   FJYHBG QA I ZIVLWU AMB WN TMBBMZA. EQTT QB ABWX GWC NZWU LMKZGXBQVO BPQA?
   ```
8. Crack the following Caesar cipher:
   ```
   ZHPSAOLZLCLUZLHZVJGPGZVUJKHVKUGZEGNNGPVALVLREXSBBJMXQEVZUOQXKPAZQ
   ```
9. Crack the following Caesar cipher:
   ```
   LXI WLDG SHXC PSXUU T GTCI DGSTG RPT HPG XHWP GSTG
   ```
10. Crack the following Caesar cipher:
    ```
    XIHVJ WMRJB FJBJW JWLRN WCYNA BRJWA DUNAX OUNPN WMJAH OJVNF QXLXW ZDNAN MVJWH UJWMB
    KNOXA NMRBJ YYNJA RWPRW CQNBJ WMBXO CRVNV DLQUR TNCQN YQJAX JQBXO NPHYC RJWCR VNB
    ```

11. Crack the following Caesar cipher:
    `UXZAPRQUMUAYLGUHWIXYZLIGQILFXQULNQI`
12. Crack the following Caesar cipher:
    `CIORCIUCIWGHVSTCFASFQODWHOZCTPIFYWBO-TOGC`
13. Crack the following Caesar cipher:
    `EZZKXHOASOYSELGBUXOZKSKZGR`
14. Crack the following Caesar cipher:
    `QYVPQUCGUHQFPGGJVVCUGKFMEQNTGJU`

Chapter 4

Combinatorics

The field of combinatorics studies the mathematics of counting. Although it is a deep mathematical field, our study is primarily based on the next result.

Fact 4.1. *Suppose that event A has a possible results and event B has b possible results, and the result of one event doesn't affect the other. Then there are ab possible results for both events combined.*

Definition 4.2. When the outcome of an event does not affect another event, the events are *independent*.

Example 4.3. Suppose we roll a red die and a blue die. Since each die has six possible results, the act of rolling both dice has 36 possible results.

We are often interested in the likelihood of a particular result. This is based on the number of ways of obtaining the given result and the number of all possible results.

Definition 4.4. The *probability* of obtaining a particular result in an event is the number of ways the result can occur divided by the number of possible results that could occur from the event. We denote the number of ways result x can occur as $N(x)$, and the probability of obtaining result x as $P(x)$.

Dice are still a good example.

Example 4.5. When rolling two distinct dice, we saw there are 36 possible combined outcomes. Since there is only one way to roll a total of two, $P(2) = 1/36$. Since it is impossible to roll a total of one, $P(1) = 0$. Since there are three ways of rolling a total of four, $P(4) = 3/36 = 1/12$.

When events do not influence each other, probabilities have many nice properties. In particular, they multiply.

Fact 4.6. *Suppose we perform events A and B independently. The probability of obtaining result x in event A and result y in event B is $P(x)P(y)$.*

Proof. Suppose event A has a outcomes and event B has b outcomes. Then $P(x) = N(x)/a$ and $P(y) = N(y)/b$. The number of ways of obtaining x and y is $N(x)N(y)$, and the number of total results for both events is ab. The result follows. □

4.1 Ordering

An important problem is combinatorics is determining the number of ways of ordering a group of objects.

Example 4.7. Suppose we want to order three people in a line. Consider the act of choosing the first person as an event, choosing the second person as an event, and also the third. Provided we account for how many people have already been chosen at each event, the events are independent. Hence, we obtain $3 \times 2 \times 1 = 6$ orderings.

This type of computation occurs so frequently that we give the descending multiplication its own symbol.

Fact 4.8. *The number of ways to order n objects is $n! := n(n-1)(n-2)\cdots 1$.*

4.2 Choosing groups

Sometimes we want to take a group of objects and pick a smaller group from it in such a way that order does not matter.

Example 4.9. Suppose we have six people in a class and want to choose three of them to form a group. Counting the number of ways to choose the first person, the second, and the third, we may be tempted to answer $6 \times 5 \times 4 = 120$. However, this assumes that order matters, which it does not. For any given group of three people, we counted each of the $3! = 6$ orderings separately, so we should divide by this number to obtain 20 possible groups.

Notice something interesting about the previous example. We could have written the result as

$$\frac{6!}{3!3!} = 20$$

instead.

Example 4.10. We can choose a group of three people from a group of eight people in

$$\frac{8!}{3!5!} = 56$$

ways.

This approach works generally, and the result is given a special symbol.

Fact 4.11. *The number of ways of choosing a group of k objects from a group of n objects, when order does not matter, is*

$$\binom{n}{k} := \frac{n!}{k!(n-k)!}$$

and is read "n choose k".

4.3 Inclusion-exclusion principle

Sometimes we wish to count a number of objects with certain properties.

Example 4.12. If there is a group of ten students and six study mathematics, how many do not study mathematics?

To count the number is easy. We simply subtract the number who do study mathematics from the total number, to obtain $10 - 6 = 4$ students.

The previous example was trivial, but what if we have more criteria that may overlap?

Example 4.13. Suppose there are ten students in a group. Suppose that five of them study mathematics, six of them study physics, and two of them study both. How would we determine the number of students studying neither?

One way to do it is the following. If we take the total number of students and subtract both the number studying mathematics and the number studying physics, we obtain $10-5-6 = -1$, which doesn't even make sense! What happened? Notice that if a student studies both subjects, she was subtracted twice. Since we only wish to subtract her from the total once, we should add back the number of students studying both subjects, to obtain $10 - 5 - 6 + 2 = 1$. Now we have correctly counted that a single student studies both subjects.

This principle, that of adding and subtracting certain sets of objects to count them correctly, is called the *inclusion-exclusion principle* and is used in many combinatorial arguments.

Example 4.14. Suppose that a certain online service requires that your password contain exactly ten characters, where a character is defined to be a lowercase letter, an uppercase letter, or a digit. However, you are required to have at least one uppercase letter and at least one digit in the password. How many passwords are possible?

To solve this, use the inclusion-exclusion principle to take the total number of passwords and correctly subtract all passwords that do not meet the criteria. Note that there are $(26 + 26 + 10)^{10} = 62^{10}$ possible sets of ten characters (since there are 26 lowercase letters, 26 uppercase letters, and 10 digits). If we first subtract all passwords that do not contain any uppercase letters, we obtain $62^{10} - 36^{10}$ since any such password contains only lowercase letters and digits. If we also remove all passwords that do not contain any digits, we obtain $62^{10} - 36^{10} - 52^{10}$ since any such password contains only lowercase and uppercase letters. However, we have twice subtracted any password that contains neither uppercase letters nor digits; these passwords must contain only lowercase letters. We must add back these 26^{10} passwords to obtain a total of $62^{10} - 36^{10} - 52^{10} + 26^{10} \approx 691$ quadrillion passwords that may be used.

What if we have more criteria than just two? The inclusion-exclusion principle may still be used, but we must be careful how it is applied.

Example 4.15. Suppose we have a group of twenty Foo monsters. Of this group, 12 have red spots, 14 have green spots, eight have blue spots, nine have red and green spots, six have green and blue spots, five have red and blue spots, and four have all three colors of spots. How many Foo monsters have no spots?

We begin by subtracting the number with each single color of spot from the total. However, we have removed those with two colors of spots too many times! We add in the number with each of two colors, but then have added the number with all spot colors one too many times! We must subtract it:

$$20 - 12 - 14 - 8 + 9 + 6 + 5 - 4 = 2$$

Hence, there are two Foo monsters with no spots.

4.4 Exercises

1. A door lock has a keypad with digits one through eight. The combination has six digits. How many possible combinations are there?
2. License plates in North Dakota have three letters followed by three numbers. How many license plates are possible?
3. A gym locker has 40 numbers on its dial, and three turns are required for each combination. If a thief had a mechanical device to check 100 combinations per second, how long would it take to try every combination? Could the thief reasonably break in?
4. If the gym locker dial is modified so it has 400 numbers on its dial, with three turns required for each combination, could the thief use his device to reasonably break in?
5. Seven people are sitting at a circular table. In how many ways can they be seated at the table if a rotation of everyone is considered the same seating arrangement?
6. In a card tournament, six people (call them Alice, Bob, Charlie, David, Ellie, and Frank) are to be split into three partnerships. Show that there are 15 ways to do this. Verify this answer by writing out the partnerships.
7. There are 26 children in a gym class. Twelve will be chosen to play catch in six pairs, while the other fourteen will play dodgeball. In how many ways can the six pairs be chosen?
8. In how many ways can you order three of the integers from one to four so that all three integers are in ascending numerical order? What about three integers from one to five?
9. You are going grocery shopping and have eight items on your list. The store might have any, all, or none of the items in stock. How many possible selections of groceries could you end up with after your trip to the store?

10. Suppose you shuffle a standard deck of playing cards (without jokers) so that the cards are in a random order. If you shuffle the cards once per second, how long will it take to see all possible orderings?

11. Suppose you reduce the size of your deck of cards to only ten cards. Repeat the same exercise.

12. Determine the number of cards required such that if you shuffled once per second, it would take approximately the length of the universe's lifespan to see all possible orderings.

13. You have six different colors of paint and want to paint a cube so each face is a different color. Notice that two paintings are the same if you can orient the cube to get from one to the other. How many different ways can you paint the cube?

14. An online service requires a password of exactly eight characters (uppercase letters, lowercase letters, and digits). The password must contain at least one letter (whether uppercase or lowercase) and at least one digit. How many passwords are possible?

15. Determine the number of ways to place n rooks on an $n \times n$ chessboard such that no rook lies in the path of another rook.

Chapter 5

Monoalphabetic substitution cipher

A *monoalphabetic substitution cipher* encrypts plaintext by swapping every occurrence of a given letter by another letter. In this case, the key to such a cipher is the encryption of each letter; that is, for every possible plaintext letter, what is the corresponding ciphertext letter?

We often note the key by a table, making it easy to see how letters are encrypted. For example, the classic Caesar shift is a simple example of a monoalphabetic substitution cipher, with this key table:

a	b	c	d	e	⋯
D	E	F	G	H	⋯

There are several important properties we need in order to devise a monoalphabetic substitution cipher. First, we must establish the corresponding ciphertext letter for each plaintext letter. Second, we must ensure that we do not repeat any letter in the ciphertext alphabet. If we did, it would not be possible to decrypt unambiguously.

To make it easier to remember and transmit the key, we often form the ciphertext alphabet using a keyword instead of a random mix of letters. To do this, we start the ciphertext alphabet with the keyword, removing double letters. After the last keyword letter, we write the next letter in the alphabet and continue, skipping any letters we already used.

Example 5.1. A monoalphabetic substitution cipher with the keyword "bookstore" has the following key table:

a	b	c	d	e	f	g	h	i	⋯
B	O	K	S	T	R	E	F	G	⋯

Definition 5.2. A *digraph* is a pair of letters, either in plaintext or ciphertext. A *trigraph* is a set of three letters. In general, an *n-graph* is a set of n letters.

If the ciphertext from a monoalphabetic substitution cipher includes spaces, it is often possible to use an analysis of language and letter repetition patterns to determine corresponding plaintext letters. That is, we can use the fact that a monoalphabetic substitution cipher encrypts n-graphs to n-graphs uniquely. For example, if the ciphertext trigraph `XUB` appears often as a ciphertext word, we might expect the mappings t $\mapsto$ `X`, h $\mapsto$ `U`, and e $\mapsto$ `B`.

5.1 Frequency analysis

What happens when a monoalphabetic substitution cipher does not include spaces in the ciphertext? This makes it difficult to perform analysis on ciphertext words directly. In this case, we can use the fact that a given plaintext letter always encrypts to the same ciphertext letter. Since some plaintext letters are very common, and others very uncommon, we should look for similar frequencies in an encrypted message. Appendix B gives the relative frequencies of letters in long English texts. To apply the table, we must determine the ciphertext letter frequencies in a given message by dividing the number of occurrences of each letter by the total number of letters in the message and multiplying by 100 to obtain a percentage.

5.2 Exercises

1. How many possible keys are there for a monoalphabetic substitution cipher?
2. How many monoalphabetic substitution cipher keys leave at least half of the plaintext alphabet unchanged by encryption?
3. Decrypt the following ciphertext using the keyword ASTLEY:

   UBJGC REOCI AKCIE AKKGE QILEN PBEOE ANCTF AOPGE W

4. Crack the following monoalphabetic substitution cipher:

   ID NQY NQY, AVII EAYHVCEB FVY. RHDLY AQ CUYLQ JD JUY ZYLQ NKJ JUY ZYLQ MEI RHQ. EBR JUYA TDDR DZR NDQI MYHY RHVBXVBT MEJYH NYCEKIY VJ VI TDDR PDH QDK.

5. Decrypt the following monoalphabetic substitution cipher using the keyword WIZARD:

   LQ. WMA LQS. AUQSKRB ND MULIRQ DNUQ OQGVRT AQGVR XRQR OQNUA TN SWB TFWT TFRB XRQR ORQDRZTKB MNQLWK, TFWMJ BNU VRQB LUZF. TFRB XRQR TFR KWST ORNOKR BNU'A RYORZT TN IR GMVNKVRA GM WMBTFGME STQWMER NQ LBSTRQGNUS, IRZWUSR TFRB HUST AGAM'T FNKA XGTF SUZF MNMSRMSR.

6. Crack the following monoalphabetic substitution cipher:

   WBB XD W VFK WDE TFGGRI JZP, KVR ABFFEP JLD, WK DFFD, IXUVK LG WAFMR KVR CWJK EXE JKWDE, DF AXUURI KVWD KVR CFFD. EWP WSKRI EWP, EWP WSKRI EWP, NR JKLTZ, DFI AIRWKV DFI CFKXFD; WJ XEBR WJ W GWXDKRE JVXG LGFD W GWXDKRE FTRWD. NWKRI, NWKRI, RMRIP NVRIR, WDE WBB KVR AFWIEJ EXE JVIXDZ; NWKRI, NWKRI, RMRIP NVRIR, DFI WDP EIFG KF EIXDZ.

7. Crack the following monoalphabetic substitution cipher:

   BEBOPBOP

8. Crack the following monoalphabetic substitution cipher:

   IB WEPI WJ ZWYP I MWHLJ, HPJWZWPBK, VWUVZR CDBKIUWDLJ. KVP JAIZZPJK JPPE DT IB WEPI CIB UHDO. WK CIB UHDO KD EPTWBP DH EPJKHDR RDL.

9. Crack the following monoalphabetic substitution cipher:

   VKEU YAU MNV RCTAWUC KC YAU TNYASIJB AMI ARFCTV, QWEVC VKC TNMVSASB; RWV HNS UNLC VELC PAUV KC KAI RCCM EM AM NXCSUVSAEMCI ESSEVARJC TNMIEVENM, XCSOEMO NM KBPNTKNMISEA. KC KAI RCTNLC UN TNLPJCVCJB ARUNSRCI EM KELUCJH, AMI EUNJAVCI HSNL KEU HCJJNYU VKAV KC ISCAICI LCCVEMO, MNV NMJB KEU JAMIJAIB, RWV AMBNMC AV AJJ. KC YAU TSWUKCI RB PNXCSVB, RWV VKC AMZECVECU NH KEU PNUEVENM KAI NH JAVC TCAUCI VN YCEOK WPNM KEL.

10. Crack the following monoalphabetic substitution cipher:

    VXIG GUV NEEO WGE QUWHEOW VXDDESUOIE PD WGEVE IPLPOHEV; UOT VXIG HV OPZ WGE OEIEVVHWB ZGHIG IPOVWSUHOV WGEM WP ULWES WGEHS DPSMES VBVWEMV PD FPYESOMEOW. WGE GHVWPSB PD WGE QSEVEOW KHOF PD FSEUW NSHWUHO HV U GHVWPSB PD SEQEUWET HOJXSHEV UOT XVXSQUWHPOV, ULL GUYHOF HO THSEIW PNJEIW WGE EVWUNLHVGMEOW PD UO UNVPLXWE WBSUOOB PYES WGEVE VWUWEV. WP QSPYE WGHV, LEW DUIWV NE VXNMHWWET WP U IUOTHT ZPSLT.

11. Crack the following monoalphabetic substitution cipher with the given frequency table:

    QGWUQ CGJQJ JCVQN QCUSW JDSDQ GWENS GQWSM YBGDC EDUQL CGEBV

BNWAV SUUJD CUJDL DQGWE NSGQW SJDQJ KCJDC JJDBM TQLUJ AVBKJ
DCGSS GSTCS UJBJC GLACJ UCGJD LTSNZ LQGWJ DSVBN WWCWE NCGQG
WJDSY SBYVS WCWXS QUJMY BGJDS VQTAU QGWUV BJDUQ GWZQN YQGWQ
GZDBI CSUQG WBNQG EMJQG UQGWA NSQFX QUJZS NSQVU QGWXN MCJAQ
JU

A	B	C	D	E	F	G	H	I	J	K	L	M
2.4	4.8	7.1	6.7	2.4	0.4	9.9	0.0	0.4	10.3	0.8	2.4	2.0

N	O	P	Q	R	S	T	U	V	W	X	Y	Z
4.8	0.0	0.0	11.9	0.0	8.7	1.6	7.1	3.6	7.9	1.2	2.0	1.6

12. Crack the following monoalphabetic substitution cipher:

BTWDR TAEIE RTMNT BJERB TWDRT AEWMQ RTMNT BJERB TWDRT AEDSE
MNWBR KMJBT WDRTA EDSEM NNMMH BRALE RRBTW DRTAE EOMCA MNIEH
BENBT WDRTA EEOMC AMNBL CQEKU HBTYB TWDRT AERED RMLMN HBSAT
BTWDR TAERE DRMLM NKDQG LERRB TWDRT AEROQ BLSMN AMOEB TWDRT
AEWBL TEQMN KEROD BQWEA DKEVE QYTAB LSIEN MQEUR WEADK LMTAB
LSIEN MQEUR WEWEQ EDHHS MBLSK BQECT TMAED VELWE WEQED HHSMB
LSKBQ ECTTA EMTAE QWDYB LRAMQ TTAEO EQBMK WDRRM NDQHB GETAE
OQERE LTOEQ BMKTA DTRMJ EMNBT RLMBR BERTD UTAMQ BTBER BLRBR
TEKML BTRIE BLSQE CEBVE KNMQS MMKMQ NMQEV BHBLT AERUO EQHDT
BVEKE SQEEM NCMJO DQBRM LMLHY

A	B	C	D	E	F	G	H	I	J	K	L	M
5.9	9.5	1.5	5.9	14.5	0.0	0.4	2.5	1.1	1.1	2.9	4.6	9.3

N	O	P	Q	R	S	T	U	V	W	X	Y	Z
4.0	2.1	0.0	5.7	8.8	2.7	10.1	1.1	1.1	4.4	0.0	0.8	0.0

13. Crack the following monoalphabetic substitution cipher:

ZUQRV ZAIEK SZRCT RZCEP YEKFP ABABA NJUBA SREZR TKSEK EJBOA

RTYIU BIYAZ HADEB ARCAZ EJKSE DTFEK FBAYA TZAZJ ZZRAG EDDHA

DEBAR CAOUK JRAUZ JGTKS YTKSO UKAZH YEPVE JBYAF ZEDDZ UQRVZ

AIEKS ZRERT WAUKR CABUK FEDRB UHJRA ZTYYA NJUSU ZRTKR DBEOR

CAIEB KJIEG UTTFU TKRFE YSAKC EBKZC TGASY UWATI EKAPU RCTIJ

BMASR TUYRC AOEJR CEDPC UICUZ TRYAT ZRRPA KRVDA ARCUF CZGUY

YUKFE MABPU RCRCA RCUKF ZRCTR PUYYF UMAJZ YUDAC ABAUK RCATB

AKT

Chapter 6

Primes and divisibility

Long division is a process familiar from school. The ideas behind the division process, notably that of quotient and remainder, have deep mathematical consequences, and form much of the underpinning to modern cryptosystems. We begin by simple division examples, introducing some common notation and terminology we will use.

6.1 Division algorithm

Definition 6.1. Suppose we divide b by a and obtain the equation $b = qa + r$. We call q the *quotient* and r the *remainder* in this operation.

Example 6.2. We can write $23 = 7 \times 3 + 2$ to represent the fact that when we divide 23 by 3, we obtain a quotient of 7 and a remainder of 2.

Example 6.3. We can write $21 = 7 \times 3 + 0$ to represent the fact that when we divide 21 by 3, we obtain a quotient of 7 and no remainder.

These examples illustrate the key components of any integer division computation. We might ask if it is always possible to obtain equations like in these examples, regardless of our choice of a and b. It turns out we can, and with suitable restrictions on the remainder, there is only one way to do so.

Fact 6.4 (Division algorithm)**.** *Let a and b be any positive integers. Then there are unique integers $q > 0$ and $0 \leq r < b$ such that $a = qb + r$.*

Proof. We will show the existence of such an equation, and leave the uniqueness as an exercise. Choose any positive integers a and b. If a is a multiple of b, then $a = qb$ for some $q > 0$, so $r = 0$ and we are done. If not, draw a number line and label the points $\ldots, -2b, -b, 0, b, 2b, \ldots$ on the line. Then a must lie strictly between two of these points; suppose $qb < a < (q+1)b$. Then $a = qb + (a - qb)$, so $r = a - qb$. Since $qb < a$, we have $r > 0$. Since $a < (q+1)b$, we have $r < q$. □

Definition 6.5. Using the notation above, suppose $r = 0$. Then we say:

- b divides a
- b is a divisor of a
- b is a factor of a
- a is a multiple of b
- a is divisible by b

We write $b|a$ in this case.

6.2 Prime numbers

It is often important to know how to decompose an integer into factors, but it is not always clear how to do so. For example, we can write $24 = 12 \times 2 = 3 \times 4$ as two ways to write 24 as a product of factors. The idea of prime numbers gives a useful and unambiguous way to do this.

Definition 6.6. We say a positive integer $p > 1$ is *prime* if its only factors are 1 and p. Integers that are not prime are *composite*.

Remark 6.7. The integer 1 is, by convention, neither prime nor composite.

Example 6.8. The integers $2, 3, 5, 7, 8675309$ are prime. The integers $6, 9, 24, 1000$ are composite.

Theorem 6.9 (Fundamental theorem of arithmetic)**.** *Every positive integer can be uniquely factored into primes, apart from the ordering of factors.*

Remark 6.10. By convention, we combine prime factors as powers and write factors in ascending order. For example, we would factor $168 = 2^3 \times 3 \times 7$.

6.3 Exercises

1. Without using a calculator, determine all prime numbers less than 100.
2. Without using a calculator, factor each of the following integers into primes.
   (a) 2013
   (b) 4140
   (c) 97
   (d) 476
3. Determine if each of the following is true or false. If true, explain why; if false, give a counterexample.
   (a) All primes greater than two are odd.
   (b) The product of two prime numbers is prime.
   (c) The sum of two prime numbers is prime.
   (d) There exist two prime numbers whose sum is prime.
4. Perform the division algorithm to divide 4096 by 29.
5. Perform the division algorithm to divide 4096 by 2.
6. Perform the division algorithm to divide 20132013 by 683.
7. Perform the division algorithm to divide -42424242 by 68.
8. Show that if a divides b and c, then a divides $b + c$. Give an example to show this works.
9. Show that if a divides b and b divides c, then a divides c. Give an example to show this works.
10. Show that if a divides $b + c$ and a divides c, then a divides b. Give an example to show this works.
11. To test if a decimal number is divisible by two, we can check if its last digit is even. Use divisibility to show that if any number of at most three digits is divisible by two, then the rule works. (Hint: write the number as $n = 100a + 10b + c$, where the digits are a, b, and c.)
12. To test if a decimal number is divisible by three, we can add its digits and check if this sum is divisible by three. Use divisibility to show that if any number of at most three digits is divisible by three, then the rule works.

13. To test if a decimal number is divisible by eleven, we can form an alternating sum of its digits (that is, look at digits from the right and alternate positive and negative terms) and check if this sum is divisible by eleven. Use divisibility to show that if any number of at most three digits is divisible by eleven, then the rule works.

14. Show the uniqueness of the division algorithm. To do this, suppose that you divide a by a such that $a = qb + r$ and $0 \leq r < b$, but a friend does the same and obtains $a = q'b + r'$ with $0 \leq r' < b$. These equations together imply that $0 = (q - q')b + (r - r')$. We want to show that $q = q'$ and $r = r'$.

    (a) Explain why $r - r'$ must be a multiple of b.

    (b) Use the given inequalities to show that $-b < r - r' < b$.

    (c) Show that $r = r'$.

    (d) Show that $q = q'$.

Chapter 7

Primality testing

A key idea in modern cryptography is that it is extremely difficult to factor large integers into primes or, alternatively, determine if an integer is already prime. When an integer n is small, we can often factor by inspection or test many known small prime numbers. But what if n has a hundred digits? Testing all possible prime factors is computationally infeasible, even for a very fast computer. A result of Fermat will help us on our quest.

Theorem 7.1 (Fermat's little theorem)**.** *Let p be a prime number. Then $a^p \equiv a \bmod p$ for every integer a.*

Example 7.2. Let $p = 5$. We have the following:

$$\begin{aligned} 0^5 &= 0 \equiv 0 \bmod 5 \\ 1^5 &= 1 \equiv 1 \bmod 5 \\ 2^5 &= 32 \equiv 2 \bmod 5 \\ 3^5 &= 243 \equiv 3 \bmod 5 \\ 4^5 &= 1024 \equiv 4 \bmod 5 \end{aligned}$$

This can provide a quick indication if a number is composite.

Fact 7.3. *If $a^n \not\equiv a \bmod n$ for any integer a, then n is composite.*

Example 7.4. Let $n = 827643$. Then $2^{827643} \equiv 8 \bmod 827643 \not\equiv 2 \bmod 827643$, so n is composite. In fact, we can factor $827643 = 3 \times 275881$.

A number a for which $a^n \not\equiv a \bmod n$ is called a *Fermat witness* for (the compositeness of) n. In the previous example, the number 2 is a witness for 827643. The previous fact tells us that finding a single Fermat witness for n tells us that n must be composite. However, the reverse is not true! It is not sufficient to choose a single integer, apply Fermat's little theorem, and conclude that a number is prime.

Example 7.5. We have $2^{341} \equiv 2 \bmod 341$, but $341 = 11 \times 31$ is not prime.

To make matters worse, there exist positive integers that are composite, but have no Fermat witnesses! Such numbers are called *Carmichael numbers* in honor of the mathematician R.D. Carmichael, who discovered their existence in 1910. It was later proven in 1984 that, while Carmichael numbers are rare, there are infinitely many of them.

Example 7.6. The integer 561 is a Carmichael number (and, interestingly, is the smallest). That is, $a^{561} \equiv a \bmod 561$ for every integer a, but $561 = 3 \times 11 \times 17$ is composite. It has no Fermat witnesses.

We need a better test. Unfortunately, there is not a known test to quickly guarantee that a number is prime. However, several probabilistic tests exist. These provide an indication that a number is likely prime, to within a desired degree of certainty. One such test is the *Miller-Rabin test*. It uses the following technical fact.

Fact 7.7. *Let $p > 2$ be a prime number which we can write as $p = 2^k q + 1$ for some $k > 0$ and odd q. If a is any integer not divisible by p, then either*

(a) $a^q \equiv 1 \bmod p$; or

(b) one of $a^q, a^{2q}, a^{4q}, \ldots, a^{2^{k-1}q} \equiv -1 \bmod p$.

Proof. Notice that in the above list, each number is the square of the preceding number, and the square of the last number is precisely $a^{2^k q} = a^{p-1} \equiv 1 \bmod p$ by Fermat's little theorem. So either the first number $a^q \equiv 1 \bmod p$ or some other number in the list is not congruent to $1 \bmod p$ but squares to become congruent to $1 \bmod p$. This second case can only occur when that number is congruent to $-1 \bmod p$. □

We can use this fact to define the idea of a Miller-Rabin witness for compositeness.

Definition 7.8. Let n be an odd integer written as $n = 2^k q + 1$ for some $k > 0$ and odd q. An integer a coprime to n is a *Miller-Rabin witness* for the compositeness of n if

(a) $a^q \not\equiv 1 \bmod n$; and

(b) $a^{2^i q} \not\equiv -1 \bmod n$ for all $i = 0, 1, \ldots, k-1$.

Essentially, this definition uses the contrapositive of the previous fact. Like the Fermat method, finding a number a that is not a Miller-Rabin witness for n does not guarantee that n is prime. In practice, the following steps are used to determine if a is a Miller-Rabin witness for n.

1. If n is even or $1 < \gcd(a, n) < n$, the number is composite. Stop the test.
2. Write $n = 2^k q + 1$ for some $k > 0$ and odd q.
3. If $a^q \equiv 1 \bmod n$, then a is not a witness. Stop the test.
4. If any of $a^q, a^{2q}, a^{4q}, \ldots, a^{2^{k-1}q} \equiv -1 \bmod n$, then a is not a witness. Stop the test if this occurs.
5. If none of these is congruent to $-1 \bmod n$, then a is a witness.

Why is this more complicated test of greater value than the Fermat test? It turns out that there are no Carmichael-style numbers for the Miller-Rabin test! Further, it can be proven that at least 75% of numbers $1 < a < n$ are witnesses if n is composite. This means that if the Miller-Rabin test is run k times and no witnesses are found, the likelihood that n is composite is at most about $1/4^k$.

Example 7.9. Suppose we want to apply the Miller-Rabin test to see if an integer n is likely prime. We run the test successfully four times. The probability that n is composite is at most about 0.39%, so there is about a 99.61% probability that n is prime. If we wanted greater certainty, we could run the test additional times.

It is common in computing to use these tests together. One such application, GNU Privacy Guard, chooses random numbers and first runs Fermat tests as a quick check for compositeness. If the number passes these tests and no Fermat witnesses are found, the program runs the Miller-Rabin test 64 times before concluding that the number is likely prime if no Miller-Rabin witnesses are found.

Most modern cryptographic algorithms rely on extremely large prime numbers, often with hundreds of decimal digits. It is natural to ask how long it takes to find such a prime. We could choose a number, run the Miller-Rabin test, and see if the number is likely prime. If it is not, we choose another number and try again. On average, how many numbers must we test before we reach one that is likely prime?

The answer lies in one of the most famous results in mathematics, the *prime number theorem*. The statement of the theorem is beyond the scope of this course, but we can state a version of it that is useful for our purposes.

Theorem 7.10 (Prime number theorem)**.** *The number of primes between x and y is approximately*

$$\frac{y}{\ln y} - \frac{x}{\ln x}$$

for positive integers $y > x$.

Example 7.11. Suppose we want to find a prime number with approximately 300 digits; that is, we want to know the number of primes p such that $10^{300} < p < 10^{301}$. The prime number theorem tells us that there are approximately

$$\frac{10^{301}}{\ln 10^{301}} - \frac{10^{300}}{\ln 10^{300}} \approx 10^{298}$$

such primes.

Another way of looking at primality testing is the following result, which follows from the prime number theorem.

Fact 7.12. *The probability that a randomly chosen number N is prime is approximately $1/\ln N$.*

Example 7.13. Suppose we are again searching for 300-digit primes. It can be shown that we must test about $\ln(10^{300}) \approx 690$ numbers of this size to find a prime.

7.1 Exercises

1. Find a Fermat witness for each of the following integers.
   (a) 9
   (b) 10
   (c) 12
   (d) 18
2. Show that 3 is not a Fermat witness for the integer 6. Why does this not contradict Fermat's little theorem?
3. If the Miller-Rabin test is run on an integer successfully six times, what is the probability that the integer is prime?
4. If the GNU Privacy Guard application runs 64 Miller-Rabin tests and concludes that an integer is prime, what is the likelihood that this is so?
5. Approximately how many times must the Miller-Rabin test be run on an integer n to give a probability that n is composite of at most 10^{-100}?
6. Using the prime number theorem, we should test approximately $\ln(10^6) \approx 14$ six-digit numbers to find a prime. Using a computer algebra system for factorization, find four different six-digit prime numbers by guessing and checking. Pay attention to how many numbers you check each time before finding a prime.

Chapter 8

Vigenère cipher

A weakness of the monoalphabetic substitution cipher is its vulnerability to linguistic techniques and frequency analysis. To combat this, we introduce the idea of a *polyalphabetic* substitution cipher. In this type of cipher, a given plaintext letter may occur in ciphertext as several different letters. Several types of these ciphers exist, but we will focus our attention on a particular example, the *Vigenère cipher*.

The key to a Vigenère cipher is a word or series of letters, which may contain repeated letters. To encrypt, repeat the keyword below the plaintext message. Apply a Caesar shift to each letter, using its numeric representation.

Example 8.1. Suppose we wish to use a Vigenère cipher to encrypt the message "cryptology rocks" using the keyword TIP. We remove spaces and repeat the keyword:

Plaintext:	c	r	y	p	t	o	l	o	g	y	r	o	c	k	s
Key:	T	I	P	T	I	P	T	I	P	T	I	P	T	I	P

We then use a Caesar shift on each letter. Since T represents a shift of 19, we have the mapping c $\mapsto$ V. Since I represents a shift of 8, we have the mapping r $\mapsto$ Z. Continuing, we obtain the following ciphertext:

Plaintext:	c	r	y	p	t	o	l	o	g	y	r	o	c	k	s
Key:	T	I	P	T	I	P	T	I	P	T	I	P	T	I	P
Ciphertext:	V	Z	N	I	B	D	E	W	V	R	Z	D	V	S	H

Notice an interesting property of the cipher. The letter o appears three times in plaintext, but is mapped to two different ciphertext letters. Similarly, the ciphertext letter V is mapped to by two different plaintext letters. This is the polyalphabetic nature of the Vigenère cipher.

A useful tool to aid in Vigenère encryption is the *Vigenère square*, an example of which appears in Appendix C. To use the square, locate the row containing the relevant key letter as its first entry. Then, find the column containing the relevant plaintext letter as its first entry. The letter where the row and column intersect is the corresponding ciphertext letter.

Example 8.2. The plaintext message "fly to the moon" encrypts to WZADFHJODCQX using a Vigenère cipher with the keyword ROCK.

To decrypt a Vigenère cipher, we simply reverse the process. Repeat the keyword beneath the ciphertext message, and reverse the Caesar shift. This can also be done using the square. Can you see how?

Example 8.3. The ciphertext DMEGFYCWIOVHHWBCSNSSQ decrypts to "a successful decryption" using a Vigenère cipher with the keyword DUKE.

8.1 Babbage-Kasiski attack

Though the Vigenère cipher appears to thwart frequency analysis and linguistic attacks because of its polyalphabetic nature, it is often possible to break. The attack, known as the *Babbage-Kasiski attack*, combines divisibility properties with frequency analysis in a clever way.

To demonstrate the applicability of divisibility to breaking the cipher, consider encrypting the plaintext message "the sun and the rain and the man in the moon" with the keyword KING. Notice that the word "the" appears four times in the plaintext message, corresponding to the three keyword trigraphs KIN, ING, and GKI; in particular, the last trigraph appears a second time. Because of this, the ciphertext corresponding to the two GKI appearances must be identical (though we don't yet know what it is). Observe that the number of positions between the two appearances is eight. Since we must have an integer number of keyword repetitions between these appearances, the length of the keyword must divide eight. This leaves several options, but narrows down the possibilities. Finding additional n-graph repetitions will narrow down our search even further.

Example 8.4. Suppose we are attempting the Babbage-Kasiski attack on Vigenère ciphertext. We find ciphertext n-graph repetitions with spacings of 8 positions, 12 positions, and 20 positions. Make a table of the factors of each spacing:

Spacing	Factors
8	1,2,4,8
12	1,2,3,4,6,12
20	1,2,4,5,10,20

Since the keyword length must be a factor of each spacing, we need a factor in common. This leaves only 1, 2, and 4 as options. We suspect the keyword likely has length four since the others seem too short to be as practical.

Once we have determined the length of the keyword, we construct frequency tables for every ciphertext letter that was encrypted with a given keyword letter. For example, suppose we know the keyword has three letters. We will have three frequency tables; the first table uses the first ciphertext letter and every third letter after, the second table uses the second ciphertext letter and every third letter after, and similarly with the third table. We can use these tables to determine what Caesar shift was used for each, and from there determine each keyword letter. Then all that remains is to decrypt the message!

8.2 Exercises

1. Decrypt the following Vigenère cipher using the keyword TROLOLO:

   BKGQFTRTPTCWOORXCEHLUXKRZKYCGWFTRLM

2. Decrypt the following Vigenère cipher using the keyword SAFETY:

   OEHEGBSNHIBDOEBEGRLOBIVYFLJEOCQOZVYPAESHLZWHNRW

3. Decrypt the following Vigenère ciphertext using the keyword ASH:

   IOHNFHBWAH WCEJFBWZTD PKWUOGUEWC EJDAK

4. Decrypt the following Vigenère ciphertext using the keyword GARY:

   ZOTYZCYRNE DYRLZQSYIC GLKCYTKMZR RGTTYCSIJK ECRSYE

5. Crack the following Vigenère ciphertext, assuming the keyword has five letters:

   LOZVI JPVXW XVZKL WLTMI FRQEK KBVUI JAPVW CFAVZ WUNFV LOMUA

   SYNCS JKAZR ZHTCW GMAKS FLVZR WMWIQ GYBRP ELVUS GTMUX GKQVS

FLNFV LOMUE JRTFV VVVYM KKIIO LOZFR WPVKL WSIEH GMUFV VVZNL

WYMKL WZPRH GDACM W

6. Crack the following Vigenère ciphertext:

UFVBRFWPEG XSAWWVTEEA HIVEKXUPPE LTHNOLRCLE FVJDIWXUDI

YIPEFPDIIB RGHACMZIAW IYEGHBOEYM GWOHRRYLIU HWRHTGULTM

IWWBPJTLML RZWVIERRZN FVDANUHRRW WBRCNLBPFW PCPMLRGDCT

AMAVWRHXZR USFRXANVBL GDEORJKMGX UHLRXXVUDK ZRZETRDUXB

QREJKGHRIH GJIECGKTUS ETULCJLTHN OLRCLFROXV ZXHGKPKXAI

LZTIIYEEPD IIBRGHACMZ IAWIYEGQNQ UFVIVRFXJI ECGKTJEFIE

HPJSGW

7. Crack the following Vigenère cipher. Once you find the keyword length, check the frequency table in Table D.1.

LLHUI ELCCP TVTGL TBBVX TKGGE ILGVR DYRRV IVTBV BHABV TWSEJ

TJHHR XVBRW IHPYM HOXHW IPQRM CZIEI SVARW IPQGV PUEHM APHLT

GVJVH TMCEX WLQBQ BVBQI ULBFI EYCZS ILHUI VLBRV PSKRP UHFRE

CKGRG JYSGL TIZRW HPBTW DMZVF TYHLX DVIEW TSJRW PURBY GWCFX

TYWGC SVCEH PPBNR SLGGE QSWFL IOWFG DUGGM IBHVS CMCEX WLIAM

ILRFX PASFS UHARV XJO

8. Crack the following Vigenère cipher. Once you find the keyword length, check the frequency table in Table D.2.

QBFIW DRRCA LHPWJ LVRUI CFBAJ WOIKD RRFSU IFKHE MOQHY SDVAC

YITAD RAOTY CWQXC BGGWF LSWGY SNIPI QTIST URXVS QPQFH AJWOI

KBRHF SGRGO UPABD TRWQK UIGHG BOCVV HVGQD QGOWE RDLRI OVSHG

RQGCQ IISQX NMUER DLRIF DTRWQ KCHPC EVDQD SUHQC U

9. Crack the following Vigenère cipher. Once you find the keyword length, check the frequency table in Table D.3.

```
QZEHA NNZEW GULKE IXHOE AHWVY FTMAT VWCIE EMFEW WSMSR SXOLG
NYHEI HLYLO XASMG LPWTB WPIDI NACUD BUFDM OHCUH BSVYU OHFEW
LEXLH YEWCL HUFON ZELSN XLOUK SOEEU EOHYT BWPIO ELKOZ LHYWA
LLHNZ EMWPU JANWA HVEKM AFKTU LIIFT IOHCU HNZEF SWMGF HSTOJ
EUFDI XNULU LWSAG DYFTC LLYLH YEAXW CYFTL WSJWC NLONZ EIHIH
AOHKO ZEAHC IHVRY IUCJE MLHUL TBWYM ZOODD XWCFS RYLHY UAOKE
MOHCU HCEPY DTBWM NGTBW SYHAL STCGN
```

10. Crack the following Vigenère cipher. Once you find the keyword length, check the frequency table in Table D.4.

```
WEISA OSKFN NGVFF TOCCI EYFIX NONHU ERLZR SAERF ODFWN FUCZP
OMDWG MEEHF WHRHV MTYWA KIEUB FYFIJ OUCRA TGVHG HIJTE OMRBL
OTYSE GUPWW USKKN NNRHR LLPCH HONWZ FEVZV NGXCG TADOX EYFIH
NDVFF TAERA EVVFT ONEOT IVVMB UUGBR VEIUB NNRZR TYFIQ OWE
```

Chapter 9

One-time pad

We have seen that patterns in Vigenère ciphertext can be used to crack messages. The Babbage-Kasiski attack uses repetitions in both the keyword and plaintext n-graphs that yield corresponding ciphertext n-graphs. To counter this attack, we might try to increase the length of the keyword to reduce this repetition. If we increase the length of the keyword to the length of the message, it is likely that no repetition will occur. This is called a *one-time pad*. The name stems from the fact that such a method is provably unbreakable provided the key is not reused for other messages; historically, such keys were written on pads of paper, one key to a sheet, and then destroyed after use. In effect, the one-time pad is a Vigenère cipher with a sufficiently long keyword.

Example 9.1. The plaintext message "onetimepadsareperfectlysecure" encrypts to

ZDLWRDRRCEKDZOEEHBIPVJPAXWZJL

using the key "lqhdjrnccbsdikpaqwencyritufsh" in a one-time pad.

9.1 Two-time pad

If a cryptographer chooses to reuse a one-time pad (effectively making it a "two-time" pad) for multiple messages, it may be possible to break all ciphertexts. If the message is standard English, it is likely the word "the" appears at least once in one of the messages. Repeat the word along one of the messages and determine the key that would be used for this encryption. Use it to decrypt the other two messages and see if any plaintext appears. If it does, continue using cribbing for this process. If not, shift the word "the" by one letter and try again.

9.2 Exercises

1. Suppose the following three messages were intercepted during the Revolutionary War, encrypted with the same one-time pad. Determine all three messages.

   XTCTSBHZWLCSCTGSSNTOWUKYDFXPVT

   QHNNFIJNWSSOTGUJRNOXFMHRESMHVP

   OERZTOHXANSASLKWELZBAOJVUSOUOV

2. You work for a super-secret intelligence agency, and are given the following order in ciphertext:

   RQXTABBJRNGGYUZGFPRE

You know it was encrypted with a one-time pad, but you don't recall which page of the pad you should use. The two remaining pages in your pad have the following keys:

LICPAQTHNVCLUDZVYVLM

MMTQAQTHNUSNRQUYNINM

What should you do?

3. You work for a super duper secret agency and receive the encrypted message `REIODUYDWDRHBXQNCOVD` from your boss. Your agency uses one-time pads to encrypt all messages, but you don't know what key was used to encrypt the message. Your pad has the two keys `lwnkdjqbskdobmqlakdl` and `mwrkdjqbsvfvxuinjkkf` on it. What action does the message instruct you to take?

Chapter 10

Euclidean algorithm

Given any integer, it is possible to factor it, often in many ways. However, it is not computationally feasible to do so for large integers. In fact, much of modern cryptography relies on the difficulty of factoring very large integers. However, we are often interested only in factors that two integers have in common and, more specifically, the greatest of these common factors.

10.1 Greatest common divisors

Definition 10.1. Given integers a and b, the *greatest common divisor* or *greatest common factor* of a and b is the largest integer that divides both a and b. We denote this integer as $\gcd(a, b)$.

Example 10.2. By listing factors and comparing, we find that $\gcd(18, 32) = 2$.

Example 10.3. By listing factors and comparing, we find that $\gcd(20, 9) = 1$.

Definition 10.4. We say a and b are *coprime* or *relatively prime* if $\gcd(a, b) = 1$.

Notice that coprime integers need not be prime. See the previous example, where neither coprime integer is prime.

10.2 Euclidean algorithm

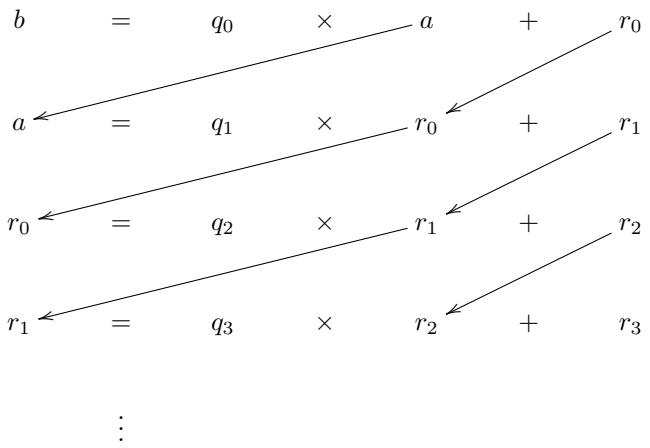
We are often interested in efficiently computing the greatest common divisor of two integers. Since it is tedious to list factors, we introduce a more efficient algorithm, the *Euclidean algorithm*. It only involves division operations. We illustrate directly with an example.

Example 10.5. Suppose we want to compute $\gcd(192, 28)$ using the Euclidean algorithm. Use the following division operations:

$$\begin{array}{ccccccccc} 192 & = & 6 & \times & 28 & + & 24 \\ 28 & = & 1 & \times & 24 & + & 4 \\ 24 & = & 6 & \times & 4 & + & 0 \end{array}$$

When we end with a remainder of zero, the previous remainder is the greatest common divisor. Therefore $\gcd(192, 28) = 4$.

The general algorithm uses the following division operations:



$$\vdots$$

The last nonzero remainder is $\gcd(a, b)$.

10.3 Extended Euclidean algorithm

For some applications, it is often not enough to know the greatest common divisor of two integers. It turns out that it is always possible to express the greatest common divisor in terms of the two original integers.

Fact 10.6 (Bézout's identity)**.** *Let a and b be any two integers. Then there are integers m and n such that $\gcd(a, b) = ma + nb$. Further, $\gcd(a, b)$ is the smallest positive integer that can be expressed in this way.*

Example 10.7. Recall that $\gcd(192, 28) = 4 = -1 \times 192 + 7 \times 28$. That is, $m = -1$ and $n = 7$.

While Bézout's identity guarantees the existence of m and n, it provides no way to determine what they are. There is an extension of the Euclidean algorithm, the *extended Euclidean algorithm*, that shows us how to find m and n. We again illustrate with an example.

Example 10.8. Recall from our earlier example that $\gcd(192, 28) = 4$. Bézout's identity guarantees the existence of m and n such that $4 = 192m + 28n$. We will start with the second-to-last equation from the Euclidean algorithm and solve each for its remainder:

$$\begin{aligned} 4 &= 28 - 1 \times 24 \\ 24 &= 192 - 6 \times 28 \end{aligned}$$

We then substitute each remainder into the equation above it, simplifying as we go (but not multiplying anything out), to obtain the following:

$$\begin{aligned} 4 &= 28 - 1 \times 24 \\ &= 28 - 1 \times (192 - 6 \times 28) \\ &= 28 - 1 \times 192 + 6 \times 28 \\ &= -1 \times 192 + 7 \times 28 \end{aligned}$$

This gives $m = -1$ and $n = 7$, as expected.

10.4 Exercises

1. Find $\gcd(-50, 40)$ using the Euclidean algorithm.
2. Find $\gcd(6337, 8053)$ using the Euclidean algorithm.

3. Find $\gcd(90210, 42)$ using the Euclidean algorithm.
4. Find $\gcd(2060478, 471339)$ using the Euclidean algorithm.
5. Carefully explain why the Euclidean algorithm must terminate with a remainder of zero.
6. Show that the result given by the Euclidean algorithm divides both integers in the algorithm.
7. Perform the extended Euclidean algorithm on the integers 54 and 19.
8. Perform the extended Euclidean algorithm on the integers 21 and 91.
9. Perform the extended Euclidean algorithm on the integers 221 and 1445.
10. Suppose that a divides bc such that $\gcd(a, b) = 1$. Use the extended Euclidean algorithm to show that a divides c.
11. Show that the result given by the Euclidean algorithm is the greatest of all numbers that divide both integers in the algorithm.

Chapter 11

Playfair cipher

Up to now, the ciphers we have examined have acted on letters individually. This allowed us to use frequency analysis with great success to break such ciphers. A way to increase the security of a cipher is to have it act on multiple letters at a time. One such cipher is the *Playfair cipher*; it acts on digraphs.

The key to a Playfair cipher is a word that has five letters after we remove any duplicates. Actually, it is possible to use keywords of any length, but this makes analysis of the cipher more difficult. We will restrict our attention to five-letter keywords. Suppose we choose the keyword CODES. Write the keyword across the top row of a 5×5 grid, and fill in the remaining alphabet, starting over with the letter nearest to A. Combine the letters I and J to ensure we do not run out of room in the grid.

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

We choose a message and break it into digraphs.

MI NI ST RY OF SI LL YW AL KS

If there are any digraphs that have a repeated letter, split them by the letter X and form digraphs again. If we end up with an odd number of letters, add an X to the end of the last unfinished digraph.

MI NI ST RY OF SI LX LY WA LK SX

The algorithm takes each digraph and replaces it with a new digraph, using the following rules:

- If the two letters are in the same row of the grid, replace each with the letter to its right, wrapping around to the start of the same row if needed.
- If the two letters are in the same column of the grid, replace each with the letter below, wrapping around to the top of the same column if needed.
- If the two letters are in neither the same row nor the same column, they define the corners of a rectangle. Replace each letter with the letter in the same row at the other end of this rectangle, maintaining the same order of letters.

In our example, we would have the following digraph encryptions, corresponding to the different rules:

- QU $\mapsto$ `RP`, BF $\mapsto$ `FG`
- DR $\mapsto$ `FX`, TY $\mapsto$ `YE`
- KY $\mapsto$ `MW`, GI $\mapsto$ `AM`

The ciphertext for our message becomes `NK IK EU TX DB CN RD MX VB ML DZ`. To disguise the fact that we used the Playfair cipher, we might remove the spaces between ciphertext digraphs, though this does not affect the encryption or decryption procedures at all.

To decrypt, we split our ciphertext into digraphs. As we will see momentarily, there is no need to split double letters. Take each digraph and replace it with a new digraph, using the reverse of the encryption rules:

- If the two letters are in the same row of the grid, replace each letter with the letter to its left, wrapping around to the end of the same row if needed.
- If the two letters are in the same column of the grid, replace each with the letter above, wrapping around to the bottom of the same column if needed.
- If the two letters are in neither the same row nor the same column, they define the corners of a rectangle. Replace each letter with the letter in the same row at the other end of this rectangle, maintaining the same order of letters.

Notice several properties of the Playfair cipher.

- It is essential that no plaintext digraph have the same letter repeated. Otherwise, the encryption rules do not apply.
- Every digraph always encrypts to the same digraph, but individual letters might encrypt in different ways.
- Playfair ciphertext, when split into digraphs, cannot have the same letter repeated within a digraph.
- A given letter can encrypt to only five different letters. A given letter can decrypt to only five different letters.

To crack a Playfair cipher when the key is not known, we could apply frequency analysis to digraphs, but there are many more digraphs that we would have to consider. Instead, we will employ a simpler cryptographic technique called *cribbing*.

Definition 11.1. A piece of plaintext believed to correspond to some piece of ciphertext is called a *crib*. The process of cracking ciphertext using a crib is called *cribbing*.

Essentially, cribbing means we use some known piece of plaintext to determine something about the key.

Example 11.2. Suppose we wish to decrypt the following Playfair ciphertext, which we have already split into digraphs:

`BO DN BI VB MK UG GQ GO EX PD CY BK YB CK QF VU LO LX LO VB OT`

When we intercepted the message, suppose we knew it was a letter addressed to Chuck. We can probably assume the letter begins with "DEAR CHUCK"; this is the crib. If our guess for the crib is correct, we already know how four plaintext digraphs encrypt:

$$\begin{aligned} \text{DE} &\mapsto \texttt{BO} \\ \text{AR} &\mapsto \texttt{DN} \\ \text{CH} &\mapsto \texttt{BI} \\ \text{UC} &\mapsto \texttt{VB} \end{aligned}$$

Look at the first digraph. There are three possibilities.

1. The letters D and E are in the same row and were shifted to the right. They could not be in the second row since `B` precedes D in the alphabet and `O` is too far ahead in the alphabet. All four letters could be in the first row, meaning they are part of the keyword.
2. The letters D and E are in the column and were shifted down. This would require D to be in the keyword, but `O` is too far ahead in the alphabet from E to have shifted.

3. The letters D and E form a rectangle and were shifted to the opposite corners of this rectangle. This means that E and O must be in the same row, so those two letters are in the keyword.

The first option is unlikely and the second impossible, so we assume the third. This leaves two possibilities for the key grid, since either the letter A is in the keyword or it is not. We can also use the third digraph mapping to determine a couple of other letters.

E		O		
B	C	D	F	G
H	I			

	E		O	
A	B	C	D	F
G	H	I		

We can immediately fill in F and G as well because of the number of spaces available. The first possibility is ruled out by the second digraph mapping, since A and D would be in the same row. This places N and R in the fourth row, and we can see where U and V likely sit. We obtain the following grid:

	E		O	
A	B	C	D	F
G	H	I		
N			R	
	U	V		

The rest of the grid is left as an exercise.

11.1 Exercises

1. Decrypt the following Playfair ciphertext using the keyword CHEERS:

   PR RS RC CU LR PR KI FU CB QF QC GU YT BI RC SL CR QD UR AU

   GI BQ PI CL PY MO CB RU OA LW

2. Decrypt the following Playfair ciphertext using the keyword SPACES:

   ZP UA FE SO MH FR MO RM SA TQ FI CN VK UG QO UA MU TX PS RO

   HU ES BV ZP LC ZC WA AU NS ME SP TI SP MK HB RY UO OF RN HU

   ES EL GQ NH UH WA IR HP

3. Use the crib DEAR CHUCK to crack the following Playfair cipher:

   BO DN BI VB MK UG GQ GO EX PD CY BK YB CK QF VU LO LX LO VB

   OT

4. Use the crib DEAR CHUCK to crack the following Playfair cipher:

   EF LI HP PF QK SV PO YR GL BV QD DH LE AD IF OV LT TU YB FZ

   EW

5. Use the crib DEAR CHUCK to crack the following Playfair cipher:

   EF MT BP PS LK AU AI KE VF QM PL KX MZ MG SP QK YA FH SL RK

   YF IG IS VI DK CS QA YF IY XK MH IA XB NT AV

6. Use the crib DEAR CHUCK to crack the following Playfair cipher:

   BV CN BI XA GQ CN IP GA CQ RO OR NP GR MQ FM QZ CN RB PO GR

   LA PE FN GZ OA RV MC SP ZQ ME KH LO HT SR GZ DX

7. Use the crib DEAR CHUCK to crack thei following Playfair cipher:

   EF OT HQ WB HO KV SZ VX XR AQ DL ZP AD QN FS SD ME LI ME WB

   ZY

8. Use the crib TO FIND A FEELING to crack the following Playfair cipher:

   AV BR MF BA GY GM RK EI OH IV PO MC OV MC DO AO RG EU SN DO

   AO RG FN RK CH GY UI DY PE UH AO IS ZF QP YD UA CH HG MT AO

   RK EQ OM GM VT DM UH TD TI LV

9. Use the crib THE VALUE AND to crack the following Playfair cipher:

   FT AZ BK ZO DK AY CU BD GF ZN HP RE DY FD MW MK OD EB ZW HL

   BH DB EO FT AZ GH BQ DK FE SN FT AZ GQ EZ UZ RY

10. Use the crib IF YOU CAN to crack the following Playfair cipher:

    LB HT CB ND AG NP CT SZ ZO MP HT XB HQ AN DY CN DY ZO YQ MP

    NV GQ IO GV

11. Use paper to make a giant Playfair key grid for working as a group, and use the crib WHERE THE to crack the following Playfair cipher:

    HE AQ AW EA DA FY ET ZN SI AW EA QA BO QE ID TL SW WK SN SI

    MX OB KM MQ AL WM LZ

Chapter 12

ADFGVX cipher

During the first World War, the German army developed a cipher, called the ADFGVX (which you might pronounce "ad-fuh-giv-ux" to save yourself some time) cipher, for use in the field. The cipher was developed by Colonel Fritz Nebel in 1918. Somewhat like the Playfair cipher, it uses a grid approach to encipher plaintext messages, but adds additional complications with the intent of befuddling cryptanalysis and cribbing.

In the keying step, we create a grid with the letters A D F G V X as the row and column headers, and insert at random the letters of the alphabet and ten digits. Unlike in the Playfair cipher, there is no need to combine letters since there are precisely enough spaces available in the key grid. The six letters of the cipher's name were chosen since they were unlikely to be confused during Morse code transmission. Below is an example of such a key grid.

	A	D	F	G	V	X
A	a	1	n	x	f	0
D	q	i	b	y	w	j
F	r	m	z	g	2	o
G	h	3	p	7	5	v
V	6	c	t	k	d	e
X	s	8	l	u	4	9

Suppose our plaintext message is "attack at 1145 hours" and we wish to use the given ADFGVX key grid. We replace each plaintext symbol with its coordinates, listing first the row and then the column. This gives the following, with coordinates written as pairs:

```
AA VF VF AA VF VG AA VF AD AD XV GV GA FX XG FA XA
```

This alone is not even remotely secure, since we have basically only developed a substitution cipher! Fortunately, we are not done. The next step is to establish a keyword with repeated letters removed, and to write the intermediate ciphertext in a grid below it. Suppose we choose SHANGHAI as our keyword:

S	H	A	N	G	I
A	A	V	F	V	F
A	A	V	D	V	G
A	A	V	F	A	D
A	D	X	V	G	V
G	A	F	X	X	G
F	A	X	A		

The columns of this grid are then rearranged into alphabetical order by the key letter:

A	G	H	I	N	S
V	V	A	F	F	A
V	V	A	G	D	A
V	A	A	D	F	A
X	G	D	V	V	A
F	X	A	G	X	G
X		A		A	F

Finally, each column is read out from top to bottom to form the final cipertext:

```
VVVXFX VVAGX AAADAA FGDVG FDFVXA AAAAGF
```

This ciphertext was then transmitted without spaces via Morse code over radio as needed.

Despite the apparent complexity of ADFGVX compared to, say, the Playfair cipher, it was quickly broken. The first successful cryptanalysis was performed by the French army lieutenant Goerges Painvin only three months after the first reported use of the cipher. Painvin used many techniques in his analysis, including cribbing, statistical analysis, repeated ciphertext, and others. However, Painvin never discovered a general technique for guaranteed decryption.

The decryption method is left to the exercises.

12.1 Exercises

1. Using the key REPORTS and the grid from the text, decrypt the following ADFGVX ciphertext: `FAAVVFXAXXXDXAGVVFDFXVXFDVDVXDFAAFVVVFXA`

2. You have intercepted the ADFGVX ciphertext `VXFXFAXFGXXVAVFAGDVFDDFVDFVXVAADFAVDFXGX` and know it was encrypted using the grid from the text. You learn from your spy network that the key has three letters, but you don't know what it is. Determine the message.

Chapter 13

Modular arithmetic

Sometimes we don't add numbers like we might expect. If the wall clock reads 10:00 right now, then four hours later it will read 2:00. In some bizarre way, we added $10 + 4 = 2$ when telling time. The concept of *modular arithmetic* makes this idea more concrete, and has huge applications for cryptology. We will use modular arithmetic and its properties extensively in later ciphers.

Suppose we start labeling the positions on a clock. The first twelve numbers are easy. If we continue, we obviously wrap around the clock, and can keep labeling. We could even go backward in time, hitting zero and then negative numbers! In this way, every integer has a position on the clock:

$$\ldots, -24, -12, 0, 12, 24, \ldots$$
$$\ldots, -23, -11, 1, 13, 25, \ldots$$
$$\ldots, -22, -10, 2, 14, 26, \ldots$$
$$\text{...and so on}$$

Basically, we put every integer into one of twelve buckets, one for each clock position. While the numbers in each buckets are certainly not all equal, there is a certain amount of "sameness" to them. Why? Notice that in any given bucket, any two numbers differ by a multiple of twelve, since we had to move around the clock a certain number of times to get from one number to the other. This leads to a definition that we will use frequently.

13.1 Equivalent integers

Definition 13.1. We say integers a and b are *equivalent* $\bmod 12$ if they differ by a multiple of 12; that is, if $12|(a-b)$. In this case, we write $a \equiv b \bmod 12$.

There is nothing special about having twelve clock positions. We could make a clock with n positions, and put numbers into buckets in the same way.

Definition 13.2. We say integers a and b are *equivalent* $\bmod n$ if they differ by a multiple of n; that is, if $n|(a-b)$. In this case, we write $a \equiv b \bmod n$.

Example 13.3. We have $35 \equiv 0 \bmod 7$ since $7|(35-0)$. Similarly, we have $-20 \equiv 4 \bmod 8$ since $8|(-20-4)$.

Notice that given any integer a, there are infinitely many integers equivalent to it $\bmod n$. It is usually most useful to deal with small positive numbers when doing computations. Therefore, we often want to find the smallest positive number equivalent to $a \bmod n$. This number is given a special name.

Definition 13.4. Let a be an integer and choose $n > 1$. The *residue of* $a \bmod n$ is the smallest positive integer that is equivalent to $a \bmod n$.

Example 13.5. The residue of $35 \bmod 3$ is 2 since this is the smallest positive integer equivalent to $35 \bmod 3$.

Note an important fact. The residue of a number $\bmod n$ is always at least zero and less than n. We will see how to efficiently calculate residues shortly.

13.2 Arithmetic

Performing modular arithmetic is particularly useful. We will illustrate with example, where we compute a modular sum in two ways.

Example 13.6. We want to simplify the sum $(134 + 25) \bmod 3$. The straightforward way is to observe that $(134 + 25) \bmod 3 = 159 \bmod 3 \equiv 0 \bmod 3$. However, observe that $134 \equiv 2 \bmod 3$ and $25 \equiv 1 \bmod 3$. If we add these residues, we obtain the equation $(134 + 25) \bmod 3 \equiv (2 + 1) \bmod 3 = 0 \bmod 3$.

This example illustrates a general fact. When we compute sums (or differences) $\bmod n$, it does not matter if we add the numbers and compute the residue, or if we compute the residues and then add them. As long as we make sure to check the final answer to ensure it is a residue, we are good to go!

The same applies to products.

Example 13.7. We want to simplify the product $(134 \times 25) \bmod 3$. One way is to observe that $(134 \times 25) \bmod 3 = 3350 \bmod 3 \equiv 2 \bmod 3$. However, observe that $(134 \times 25) \bmod 3 \equiv (2 \times 1) \bmod 3 = 2 \bmod 3$.

This example illustrates a similar fact as above. When we compute products $\bmod n$, it does not matter when we compute the residues, as long we check the final answer to ensure it is a residue.

13.3 Modular inverses

What about modular division? The answer is not as simple as for sums, differences, or products. Here is an example of why we must be extra careful in this case.

Example 13.8. We have $3 \times 2 \equiv 6 \bmod 24$ and $15 \times 2 = 6 \bmod 24$. Hence $3 \times 2 \equiv 15 \times 2 \bmod 24$. If we try to divide, we obtain $3 \equiv 15 \bmod 24$, which is false!

Let's consider what happens in ordinary integer division. You may have heard of the *reciprocal* of a number. For example, the reciprocal of 2 is $\frac{1}{2}$. We also call $\frac{1}{2}$ the *inverse* of 2, since multiplying them gives $2 \times \frac{1}{2} = 1$. Let's extend this idea to modular arithmetic, since we know how to perform modular multiplication.

Definition 13.9. We say the number a is *invertible* $\bmod n$ if there is some number b with the property that $ab \equiv 1 \bmod n$. In this case, we write $b \equiv a^{-1} \bmod n$.

Notice how this definition matches well with our original definition from above. Before learning how to compute inverses, we examine a few examples.

Example 13.10. The inverse of $3 \bmod 5$ is 2, since $3 \times 2 = 6 \equiv 1 \bmod 5$. We might write $2 \equiv 3^{-1} \bmod 5$.

Example 13.11. There is no inverse of $2 \bmod 6$. To see this, examine all possible multiplications:

b	$2b \bmod 6$
0	0
1	2
2	4
3	0
4	2
5	4

No value of b gives $2b \equiv 1 \bmod 6$, so the inverse does not exist.

It is natural to ask what conditions guarantee that $a^{-1} \bmod n$ exists.

Fact 13.12. *The inverse $a^{-1} \bmod n$ exists if and only if a and n are coprime.*

Proof. Suppose first that $a^{-1} \bmod n$ exists. That is, there is some integer m such that $am = 1 \bmod n$. By definition, we can write $am - 1 = qn$ for some integer q. Rearranging, we obtain $am - qn = 1$. Bézout's identity ensures that $\gcd(a, n) = 1$ since this is the smallest positive integer representable as a combination of a and n in this way.

Conversely, suppose that $\gcd(a, n) = 1$. Then we can write $ma + n'n = 1$ for some integers m and n'. Reducing modulo n immediately gives that $ma = 1 \bmod n$, so m is the desired inverse. □

Example 13.13. This agrees with our earlier examples. Indeed, $\gcd(3, 5) = 1$ and $\gcd(2, 6) = 2$.

To compute inverses, we use an algorithm we have already seen, the extended Euclidean algorithm. If $\gcd(a, n) = 1$, we are guaranteed integers x and y (we use these letters so they don't conflict) such that $ax + ny = 1$. If we reduce both sides of this equation $\bmod n$, we obtain $ax \equiv 1 \bmod n$ since $ny \equiv 0 \bmod n$. By definition $a^{-1} = x$. That is, once we apply the extended Euclidean algorithm, the inverse of a is the coefficient that appears with a in the algorithm.

Example 13.14. Since $\gcd(3, 5) = 1$, we are guaranteed that $3^{-1} \bmod 5$ exists. Since the extended Euclidean algorithm gives $1 = 2 \times 3 - 1 \times 5$, we have $3^{-1} \equiv 2 \bmod 5$ as expected.

13.4 Computing residues

How do we find residues? There are several ways. To calculate the residue of $a \bmod n$:

- Add or subtract multiples of n until you end up with the smallest positive result.
- Divide a by n and use the remainder.
- Guess a number and check it using the definition.

The remainder method is often the most practical. Why does it work? Suppose we apply the division algorithm to divide a by n. We obtain an equation $a = qn + r$ for some integer q and remainder $0 \leq r < n$. Since $qn \equiv 0 \bmod n$, we have $a \equiv r \bmod n$. Since $0 \leq r < n$, we must have that r is the residue of $a \bmod n$.

13.5 Solving modular equivalences

It will be useful to solve modular algebraic equivalences. The techniques are similar to those used in standard algebra, but we must be careful not to rely too heavily on our intuition.

Solving equivalences using modular addition and subtraction works just like in standard algebra.

Example 13.15. If $9 + x \equiv 2 \bmod 13$, then we subtract to obtain $x \equiv 2 - 9 \bmod 13 \equiv -7 \bmod 13 \equiv 6 \bmod 13$. Notice that we reduce to the proper residue at the end of the computation.

Example 13.16. If $8 \equiv x - 21 \bmod 134$, then we add to obtain $23 \equiv x \bmod 134$, which we typically rearrange to read $x \equiv 23 \bmod 134$. Note we already have a residue, so there is no need to reduce further.

Solving equivalences involving modular multiplication requires a bit of care. If we were to solve the standard equation $2x = 6$, we would divide both sides of the equation by 2 to obtain $x = 3$. However, division doesn't make sense in modular arithmetic; instead, it is replaced by multiplicative inverses.

Example 13.17. To solve the equivalence $2x \equiv 6 \bmod 13$, we need to cancel the 2 on the left side. Since we can't divide, we instead find $2^{-1} \bmod 13$. The extended Euclidean algorithm (or some lucky guessing) gives that $2^{-1} \equiv 7 \bmod 13$. Then the original equivalence becomes $2^{-1} \times 2x \equiv 2^{-1} \times 6 \bmod 13$. Since $2^{-1} \times 2 \equiv 1 \bmod 13$ by definition, we have

$$\begin{aligned} x &\equiv 2^{-1} \times 6 \bmod 13 \\ &\equiv 7 \times 6 \bmod 13 \\ &\equiv 42 \bmod 13 \\ &\equiv 3 \bmod 13 \end{aligned}$$

and have solved for x. This checks out, since substituting $x = 3$ into the original equivalence works.

Recall that $a^{-1} \bmod n$ only exists when a and n are coprime. What does this mean for modular algebra? If we want to solve an equivalence of the form $ax \equiv b \bmod n$ for x when $\gcd(a, n) \neq 1$, there are two possibilities:

- The equivalence has no solutions.
- The equivalence has more than one solution.

We will examine each possibility with an example.

Example 13.18. Suppose we want to solve the equivalence $3x \equiv 6 \bmod 12$ for x. Since $\gcd(3, 12) = 3 \neq 1$, we can't use an inverse to solve the equation directly. Instead, construct a table of possible values:

x	$3x \bmod 12$
0	0
1	3
2	6
3	9
4	0
5	3
6	6
⋮	⋮

Notice that setting $x = 3$ or $x = 6$ solves the equation (and, in fact, there are other solutions). Hence, the equivalence has multiple solutions.

Example 13.19. Suppose we want to solve the equivalence $2x \equiv 1 \bmod 4$. Again, we can't use inverses here. Construct a table:

x	$2x \bmod 4$
0	0
1	2
2	0
3	2

No value of x makes the equivalence true, so it has no solutions.

13.6 Exercises

1. Without using a calculator, reduce each of the following to the correct residue.
   (a) $34 \bmod 9$
   (b) $34 \bmod 10$
   (c) $-128 \bmod 24$
   (d) $-20 \bmod 7$
   (e) $8675309 \bmod 5$
   (f) $12345678 \bmod 2$
2. Use modular arithmetic in two ways to simplify each expression to the correct residue.
   (a) $(999 + 84) \bmod 5$
   (b) $(44 - 14) \bmod 3$
   (c) $(26 + 26 + 26 + 26 + 26) \bmod 13$
   (d) $(12 - 199) \bmod 70$

3. Without using a calculator or the extended Euclidean algorithm, find the given modular multiplicative inverse.
   (a) $3^{-1} \bmod 29$
   (b) $10^{-1} \bmod 29$
   (c) $3^{-1} \bmod 5$
   (d) $3^{-1} \bmod 6$
4. Use the extended Euclidean algorithm to find the given modular multiplicative inverse. Verify your answer is correct.
   (a) $54^{-1} \bmod 19$
   (b) $19^{-1} \bmod 54$
   (c) $1445^{-1} \bmod 221$
   (d) $137^{-1} \bmod 255$
   (e) $24^{-1} \bmod 80$
5. Show that $(n-1)^{-1} \equiv (n-1) \bmod n$ for any n by multiplying and finding the appropriate residue.
6. Solve each of the following equations for x.
   (a) $3x \equiv 4 \bmod 10$
   (b) $7x - 1 \equiv 0 \bmod 3$
   (c) $7x \equiv 1 \bmod 4$
   (d) $13x \equiv 1 \bmod 26$
7. Notice that $\gcd(3, 15) \neq 1$, so $3^{-1} \bmod 15$ does not exist. Verify this by computing each possible multiplication.
8. Compute $n^{-1} \bmod 29$ for all residues n. Mark your results in the table in Appendix E.

Chapter 14

Affine cipher

14.1 Additive ciphers

The *affine cipher* generalizes the Caesar shift using modular arithmetic. Before we introduce it, let's examine the Caesar shift again, this time using modular arithmetic. We first assign each letter in the alphabet a number, using the following scheme:

$$\begin{aligned} \mathrm{a} &= 0 \\ \mathrm{b} &= 1 \\ &\vdots \\ \mathrm{z} &= 25 \end{aligned}$$

The reason for beginning our count at zero will be apparent in a moment.

Fact 14.1. *If the numeric equivalent of a Caesar shift is b, then a ciphertext letter C can be determined from its corresponding plaintext letter P using the formula $C \equiv P + b \bmod 26$.*

Notice that this modular addition accounts for wrapping around the alphabet, and justifies the reason for beginning our count at zero to coincide with the start of the alphabet. The corresponding decryption formula is $P \equiv C - b \bmod 29$. Because of this formulation, the Caesar shift is sometimes called an *additive* cipher.

14.2 Multiplicative ciphers

Since the Caesar shift uses modular addition, we might wonder if a similar formulation is possible using modular multiplication. The answer is yes, but we must be careful when doing so. Instead of adding a fixed number to a plaintext letter to get the corresponding ciphertext letter, a *multiplicative* cipher would multiply some fixed number a by the plaintext letter instead. Let's examine some examples.

Example 14.2. Suppose we want to construct a multiplicative cipher with key $a = 7$. Since g $= 6$, we have the mapping g $\mapsto$ `Q` under multiplication.

Example 14.3. Suppose we want to construct a multiplicative cipher with key $a = 13$. Since d $= 3$, we have the mapping d $\mapsto$ `N` under multiplication. However, we also have the mapping h $\mapsto$ `N`.

The previous example shows that some keys result in multiple plaintext letters mapping to the same ciphertext letter. This is unacceptable, since unambiguous decryption is not possible. Why does this fail? To see, notice that the encryption formula for a multiplicative cipher is $C \equiv aP \bmod 26$. To solve for P, we would need to "divide" by a. However, division is not allowed in modular arithmetic! Instead, multiply both sides of the equivalence by a^{-1} on the left and rearrange terms to obtain $P \equiv a^{-1}C \bmod 26$. This decryption formula only makes sense if a has an inverse mod 26. However, since $\gcd(13, 26) = 13 \neq 1$, there is no inverse.

There is clearly a problem. Not every key can be used in a multiplicative cipher in this way. To avoid the problem, we often construct a new plaintext alphabet with 29 letters. Since 29 is prime, every nonzero number less than 29 is coprime to it, and therefore is an acceptable multiplicative key. This alphabet is the following, where _ denotes a space:

$$\begin{aligned} \text{a} &= 0 \\ \text{b} &= 1 \\ &\vdots \\ \text{z} &= 25 \\ \_ &= 26 \\ , &= 27 \\ . &= 28 \end{aligned}$$

14.3 Affine cipher

The affine cipher combines the additive and multiplicative ciphers, using two keys. The first key is a multiplicative key a, and the second is an additive key b. The encryption formula (using the 29-letter alphabet) becomes $C \equiv aP + b \bmod 29$. The decryption formula is left as an exercise.

14.4 Exercises

1. Determine the affine decryption formula.
2. Decrypt the following affine ciphertext, encrypted using $a = 5$ and $b = 19$:

   B_TZCQKB_,ZKURCD_F,ZKRKQBIKFTZCYYB,

3. Decrypt the following affine ciphertext, encrypted using $a = 17$ and $b = 22$:

   JX.SDY_ZDKDWVJWYJ.LGYY.O,WLQVDCDL_GWLYWVPJTNLINLI

4. The following ciphertext (known to begin with the plaintext STAR) was encrypted using an affine cipher. Decrypt the message:

   JTD.NAXQUPDTE,L,APOD.R,TPZRFNATTPO.DZR

5. The following ciphertext (known to begin with the plaintext THIS) was encrypted using an affine cipher. Decrypt the message:

   WDHSVHSVDCFVHWVFCOPSZVHWVYUUTSVEVTHWWTUVFCOSUVWDE,VFDU,

   VFUVQOCBUVC_OVDUEOSUVOH.DWVWDOC_.DVWDEWVSMOUEXH,.VMOCFQA

6. If we use a 26-letter alphabet, how many possible keys are there for an affine cipher?
7. If we use a 29-letter alphabet, how many possible keys are there for an affine cipher?

Chapter 15

Group theory

Many ciphers, including some of those we have seen so far, involve processes that share common characteristics. For example, in the Caesar shift, there is an idea of processing a plaintext letter to obtain a ciphertext letter. Decryption, as we saw, is simply the reverse operation. And we saw that a "zero shift" keeps all plaintext letters unchanged.

These simple operations are an example of what constitues a mathematical structure called a *group*. Groups play a pivotal role in many aspects of mathematics, and we will see examples of group structures throughout this course. Being able to identify group structures will be useful, since it allows us to immediately use general results in those circumstances to help us in our work.

Let's formally define what we mean.

Definition 15.1. A *group* is a set of elements along with an operation (say, $\oplus$) that combines elements such that the following are true.

- For elements x and y in the set, $x \oplus y$ is another element in the set.
- For elements x, y, and z in the set, we have $(x \oplus y) \oplus z = x \oplus (y \oplus z)$; that is, the operation is associative.
- There is an element e in the set such that $e \oplus x = x \oplus e = x$ for every element x in the set. We call the element e the *identity* element.
- For every element x in the set, there is another element y in the set such that $x \oplus y = y \oplus x = e$. We call the element y the *inverse* of x and often write $y = x^{-1}$.

Note that the definition requires that we identify both the set of elements and the operation. While this definition may seem arbitrary, it matches our intuition.

Example 15.2. The set of integers is a group under the usual addition operation. Let's verify the requirements.

- For any two integers x and y, the sum $x + y$ is another integer.
- Associativity holds; we know this already!
- The identity element is $e = 0$. For any integer x, we have $x + 0 = 0 + x = x$.
- For any integer x, the inverse is $x^{-1} = -x$ since $x + (-x) = -x + x = 0$. Note that $-x$ is also an integer.

Two different sets may be groups under the same operation.

Example 15.3. The set of even integers is a group under addition.

- For any two even integers x and y, the sum $x + y$ is another even integer.
- Associativity holds; we know this already!

- The identity element is $e = 0$. For any even integer x, we have $x + 0 = 0 + x = x$.
- For any even integer x, the inverse is $x^{-1} = -x$ since $x + (-x) = -x + x = 0$. Note that $-x$ is also an even integer.

All requirements must hold in order to have a group structure.

Example 15.4. The set of odd integers is not a group under addition since the sum of two odd integers is an even integer. It is not necessary to check the other requirements since the first does not hold.

15.1 Exercises

1. Show that the set of rational numbers is not a group under multiplication.
2. Show that the set of nonzero rational numbers is a group under multiplication.
3. Show that the set of integers that are multiples of five is a group under addition.
4. Show that the set of integers of the form 2^x (where x is an integer) is a group under multiplication.
5. Show that the set of Caesar shifts is a group, where the operation combining two shifts is the resulting composite shift.
6. We often write $\mathbb{Z}_n$ to mean the set of integers modulo n. Show that $\mathbb{Z}_n$ is a group under addition.
7. Show that $\mathbb{Z}_n$ is not a group under multiplication.

Chapter 16

Enigma cipher

The Enigma cipher machine is arguably the most famous cipher machine in history. It was used during the second World War to encrypt and decrypt German field messages. Electrical signal used for encryption and decryption followed this path:

1. The *keyboard* looked like a typewriter keyboard. Message letters were typed on the keyboard.
2. The *plugboard* looked like a telephone operator's switchboard. It was used to swap letters.
3. There were three *rotors* that scrambled the electrical signal in the machine. The rotors moved after each letter.
4. The *reflector* caused the electrical signal exiting from the rotors to enter them at a different location.
5. After encryption or decryption, a lamp on the *lampboard* lit up to indicate the resulting letter.

The machine had to be set before messages could be encrypted or decrypted. Keys consisted of three components:

1. The *plugboard setting* selected cables to connect letters on the plugboard to swap. Some Enigma models had six cables, while others had more.
2. The *rotor order* determined what order to place the rotors in the machine.
3. The *rotor orientation* determined what position to rotate each rotor initially.

A paper simulator of an Enigma machine is included in Appendix F.

Example 16.1. We might set our Enigma machine to plugboard 2 (corresponding to six pairs of swapped letters), rotor order 231, and rotor orientation AJF.

Example 16.2. Using plugboard 1, rotor order 321, and rotor orientation FAR, the plaintext message "hello" encrypts to `RIBQZ` using our Enigma simulator.

There are important properties of the Enigma cipher to consider.

1. No letter can encrypt to itself under any key. This is a consequence of the reflector.
2. The cipher is periodic. That is, it must eventually repeat if the same key is pressed. This is a consequence of the rotor rotation pattern.

To avoid repetition in ciphertext, the German military established protocols to change keys. The first was to change the global key used by operators each day. This master key was called the *day key*. Operators had a book containing the settings for the current month.

Once an operator set his Enigma machine to the day key, he would come up with three random letters and entered them twice in the machine (e.g. TIPTIP) for redundancy. This set of three letters formed the

message key. Once he entered the message key twice, he reset the rotor orientation to the message key before beginning the encryption of a message.

The receiving operator would set his Enigma machine to the day key as well, and would enter the first six ciphertext letters into the machine. This would reveal the message key and its repeat. He would reset his machine to this message key and continue decrypting the message. In this way, only six letters per message were ever encrypted with a given day key.

16.1 Exercises

1. Decrypt the following Enigma ciphertext using the day key 231 POO PL-4: `NTAGBQQWA`
2. Decrypt the following Enigma ciphertext using the day key 132 RED PL-1: `JCTYCRLKHDHIPNSVNPXZFR`
3. Decrypt the following Enigma ciphertext using the day key 321 HUG PL-2: `HRNANUYSWLGFSDC`
4. Decrypt the following Enigma ciphertext using the day key 123 AAA: `AYAYZZZMTOXMXJUQHYN`
5. Decrypt the following Enigma message, encrypted with plugboard 4, rotor order 213, and rotor orientation CTY: `MHQIGGWAGRAIDOOXZBHMAWM`
6. Decrypt the following Enigma message, encrypted with plugboard 1, rotor order 321, and rotor orientation XZZ: `IWQNRJDLSVWJZPIVXEVB`
7. Decrypt the following Enigma message, encrypted using a message key with plugboard 5, rotor order 123, and rotor orientation SKL: `PUPUQXBXOFJJHKZLS`
8. This exercise will demonstrate the importance of the plugboard. On your Enigma simulator, choose any plugboard, rotor position, and rotor orientation.
   (a) Encrypt your name. Make sure at least one letter from the plugboard is used. If not, choose another plugboard and try again.
   (b) Remove the plugboard and reset the rotors to the same starting orientation as before. Encrypt your name again.
   (c) You now have two ciphertexts, one using a plugboard and one without using a plugboard. Can one be obtained from the other by simply swapping letters according to the plugboard setting?
9. In one version of the Enigma machine, six pairs of letters are chosen on the plugboard. In how many ways can these pairs be chosen?
10. In another version of the Enigma machine, ten pairs of letters are chosen on the plugboard. In how many ways can these pairs be chosen?
11. In one version of the Enigma machine, three rotors are arranged in any order. In how many ways can the rotors be arranged?
12. In another version of the Enigma machine, three rotors are chosen from a pool of five and are arranged in any order. In how many ways can the rotors be chosen and arranged?
13. In the Enigma machine, each of the three rotors is set to any letter for its starting orientation. How many starting orientations are there for the machine?
14. Suppose we have an Enigma machine that uses (as in the previous exercises) six plugboard letter pairs, three rotors in any order, and any starting rotor orientation. These settings form the key for the machine. How many keys are possible on this machine?

Chapter 17

Matrix algebra

For many applications in science and mathematics, it is convenient to create two-dimensional grids of numbers. These grids, which we call *matrices* (the singular form is *matrix*) have no special powers on their own, but we will see that they make some ciphers possible.

Definition 17.1. A $2 \times n$ *matrix* (read "2 by n") is a rectangular grid of numbers with two rows and n columns.

Example 17.2. The grid

$$\begin{pmatrix} 1 & 0 \\ \pi & 42 \end{pmatrix}$$

is a 2×2 matrix.

Example 17.3. The grid

$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

is a 2×5 matrix.

By convention, we typically identify matrices using capital letters. It is possible to define matrices of other sizes, but we do not consider them in this course. See any textbook on linear algebra for details.

There are two special matrices that we will see frequently.

Definition 17.4. The 2×2 *identity matrix* is the matrix $I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$.

Definition 17.5. The $2 \times n$ *zero matrix* is the matrix $0 = \begin{pmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \end{pmatrix}$.

17.1 Matrix operations

It is possible to define operations, like addition and multiplication, on matrices under certain circumstances. The first is matrix addition. It operates just as we might expect.

Definition 17.6. Let A and B be $2 \times n$ matrices. The sum $A + B$ is the $2 \times n$ matrix formed by adding corresponding entries.

Example 17.7. If $A = \begin{pmatrix} 1 & 2 & 3 \\ -1 & -2 & -3 \end{pmatrix}$ and $B = \begin{pmatrix} -2 & 1 & 4 \\ 0 & 10 & 4 \end{pmatrix}$, then $A + B = \begin{pmatrix} -1 & 3 & 7 \\ -1 & 8 & 1 \end{pmatrix}$.

Note that matrix addition is only defined if the two matrices have the same size. Otherwise, it does not make sense.

It is also possible to multiply matrices. We might hope that matrix multiplication operates by multiplying corresponding entries, in the same fashion as addition. However, this is not the case, for reasons that are beyond the scope of this course. In fact, it is generally not possible to multiply two matrices of the same size at all. However, if the matrices are of the appropriate size, it is possible.

Definition 17.8. If

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$$

is a 2×2 matrix and

$$B = \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \end{pmatrix}$$

is a $2 \times n$ matrix, the product

$$AB = \begin{pmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} & \cdots & a_{11}b_{1n} + a_{12}b_{2n} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} & \cdots & a_{21}b_{1n} + a_{22}b_{2n} \end{pmatrix}$$

is a $2 \times n$ matrix.

Admittedly, this notation looks horrible. But it isn't that bad! The entry in row i and column j of the product AB is formed by multiplying and adding entries along row i of A and down column j of B. With practice, this operation becomes easy.

Example 17.9. Let $A = \begin{pmatrix} 1 & 2 \\ 3 & 0 \end{pmatrix}$ and $B = \begin{pmatrix} 2014 & 1 & 4 \\ -1 & 0 & 2 \end{pmatrix}$. Then $AB = \begin{pmatrix} 2012 & 1 & 8 \\ 6042 & 3 & 12 \end{pmatrix}$.

Remark 17.10. If A is a 2×2 matrix and B is a $2 \times n$ matrix, then AB is defined but BA is not, in general. If B is also 2×2, then the product BA is defined. However, in general $AB \neq BA$. An example is left as an exercise.

One final matrix operation is that of multiplying a number by a matrix. This works just like we might expect.

Definition 17.11. If A is a $2 \times n$ matrix and k is any number, then kA is the $2 \times n$ matrix formed by multiplying k by every entry in A.

17.2 Matrix inverses

Division of matrices, like modular division, is not defined. However, we can form the idea of a *matrix inverse* that matches the definition for modular algebra quite well.

Definition 17.12. Let A be a 2×2 matrix. We say A is *invertible* if there is some 2×2 matrix B such that $AB = BA = I$. In this case, we say B is the *inverse* of A and write $B = A^{-1}$.

Example 17.13. Let $A = \begin{pmatrix} 4 & 15 \\ 1 & 4 \end{pmatrix}$. Then $A^{-1} = \begin{pmatrix} 4 & -15 \\ -1 & 4 \end{pmatrix}$ since $AA^{-1} = A^{-1}A = I$.

If A is not a 2×2 matrix, the inverse is not defined. Technically, we can define inverses for some other matrices, but we will not consider these in this course.

Not every matrix is invertible. For example, the 2×2 zero matrix is not invertible since $A0 = 0A = 0$ for any 2×2 matrix A. How do we tell if a given matrix is invertible? There is a way to assign to every 2×2 matrix a number that gives information about the invertibility of the matrix.

Definition 17.14. Let $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$ be a 2×2 matrix. Then the *determinant* of A is the number $\det A = ad - bc$.

Example 17.15. Let $A = \begin{pmatrix} 1 & -2 \\ 3 & 4 \end{pmatrix}$. Then $\det A = 1 \times 4 - (-2) \times 3 = 10$.

The reason for defining a matrix determinant is primarily for the following fact.

Fact 17.16. *A matrix A is invertible if and only if* $\det A \neq 0$. *If A is invertible, then* $A^{-1} = \frac{1}{\det A} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix}$.

Example 17.17. The matrix $A = \begin{pmatrix} 4 & -8 \\ 1 & -2 \end{pmatrix}$ is not invertible since $\det A = 0$.

17.3 Matrix equations

One reason for introducing matrix inverses is to help solve matrix equations, much like modular inverses helped solve modular equivalences. Equations involving matrix addition and subtraction can be solved just like their standard algebraic counterparts.

Example 17.18. The matrix equation $A + X = B$, where A, X, and B are $2 \times n$ matrices, can be solved for X to obtain $X = B - A$.

Note that we need the matrices to be of the same size for the addition or subtraction to make sense. In the case of equations involving matrix multiplication, we need to use inverses.

Example 17.19. The matrix equation $AX = B$, where A is an invertible 2×2 matrix and X and B are $2 \times n$ matrices, we can multiply both sides of the equation on the left by A^{-1} and, since $A^{-1}AX = IX = X$, the equation has the solution $X = A^{-1}B$.

There are a few key points to consider with this example:

- If A is not invertible, the equation has no solutions.
- Since we cannot change the order of multiplication, it is important to multiply both sides of the equation on the same side! That is, the solution is not BA^{-1}, which may be a different matrix altogether.

17.4 Modular matrices

We can combine modular arithmetic and matrix algebra together in ways that will prove useful. A *modular matrix* is simply a matrix whose entries are taken $\bmod n$ for some number n.

Example 17.20. We have $\begin{pmatrix} 20 & -1 \\ 4 & 32 \end{pmatrix} \equiv \begin{pmatrix} 0 & 9 \\ 4 & 2 \end{pmatrix} \bmod 10$.

Algebra on modular matrices is identical to algebra on standard matrices, except that answers must be reduced $\bmod n$. The only difference occurs when we consider invertibility.

Fact 17.21. *A modular matrix* $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$ *is invertible* $\bmod n$ *if and only if* $\gcd(\det A, n) = 1$. *If* A *is invertible, then* $A^{-1} = (ad - bc)^{-1} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix} \bmod n$.

17.5 Exercises

1. Compute the following matrix product:

$$\begin{pmatrix} 7 & 25 \\ 20 & 13 \end{pmatrix} \begin{pmatrix} 42 & 42 \\ 4 & 2 \end{pmatrix}$$

2. Find two nonzero matrices whose product is zero.
3. Find two matrices A and B such that $AB \neq BA$.
4. Compute the following matrix product:

$$\begin{pmatrix} 1 & 2 \\ 3 & -4 \end{pmatrix} \begin{pmatrix} 0 & 1 & 9 & 14 & -2 \\ 4 & 1 & 2 & 0 & -1 \end{pmatrix}$$

5. Find the determinant of each real matrix:
   (a) $\begin{pmatrix} -1 & 8 \\ 4 & -5 \end{pmatrix}$

(b) $\begin{pmatrix} 4 & 8 \\ -2 & -4 \end{pmatrix}$

6. Find the inverse of each real matrix if it exists:

   (a) $\begin{pmatrix} 7 & 2 \\ 10 & 3 \end{pmatrix}$

   (b) $\begin{pmatrix} 17 & 13 \\ 10 & 8 \end{pmatrix}$

   (c) $\begin{pmatrix} 5 & 4 \\ 10 & 8 \end{pmatrix}$

   (d) $\begin{pmatrix} 2 & 4 \\ 4 & 5 \end{pmatrix}$

7. Compute the following matrix product:

$$\begin{pmatrix} 4 & 15 \\ 8 & 9 \end{pmatrix} \begin{pmatrix} 5 & 12 \\ 1 & 6 \end{pmatrix} \bmod 17$$

8. Let $M \equiv \begin{pmatrix} 12 & 8 \\ 11 & 6 \end{pmatrix} \bmod 13$. Use the fact that $10^{-1} \equiv 4 \bmod 13$ to find M^{-1}. Verify by multiplying that $MM^{-1} \equiv I \bmod 13$ and that $M^{-1}M \equiv I \bmod 13$.

9. Find the inverse of each modular matrix if it exists, and verify your answer is correct by multiplication.

   (a) $\begin{pmatrix} 7 & 2 \\ 10 & 3 \end{pmatrix} \bmod 13$

   (b) $\begin{pmatrix} 13 & 0 \\ 91 & 1 \end{pmatrix} \bmod 26$

   (c) $\begin{pmatrix} 2 & 4 \\ 4 & 5 \end{pmatrix} \bmod 5$

   (d) $\begin{pmatrix} 17 & 13 \\ 10 & 8 \end{pmatrix} \bmod 29$

10. Solve the matrix equation $AB = BC$ for C, where each matrix is a 2×2 invertible matrix.

11. Solve the matrix equation $C = KP$ for P, where K is a 2×2 invertible matrix.

12. Show the following facts about 2×2 matrices. These facts show that the set of such matrices is a group under addition.

    (a) The sum of any two 2×2 matrices is a 2×2 matrix.

    (b) The order in which 2×2 matrices are added does not matter.

    (c) There is a 2×2 matrix that, when added to a matrix A, leaves A unchanged.

    (d) For any 2×2 matrix A, there is a matrix B such that $A + B = 0$.

Chapter 18

Hill cipher

We will now use our matrix expertise to tackle a new cipher, the *Hill cipher*. First, we need a way to represent a plaintext message as a matrix:

1. Translate each plaintext letter in the message to its numerical value in the 29-letter alphabet.
2. Insert the message into a $2 \times n$ matrix by entering the numerical values down the first column, then the second, and so on. The number of columns will depend on the length of the message.
3. If the last numerical value in the message ends in the first row of a column, add a dummy value to the end, like the numerical value for the letter x or a space.

We could use the 26-letter alphabet, but it will become clear that this is less useful than the 29-letter alphabet.

To encrypt a message once it has been converted to a plaintext matrix P, we choose a key matrix K. This is a 2×2 invertible matrix of our choice. Since we require K to be invertible, it is most useful to use a 29-letter alphabet, since invertibility $\bmod 29$ is easier to achieve than working $\bmod 26$. We form a $2 \times n$ ciphertext matrix C according to the formula $C \equiv KP \bmod 29$. Note that this multiplication makes sense since the matrices are of the correct size.

Example 18.1. Suppose we want to encrypt the message "hello" using the Hill cipher. We choose a key matrix $K = \begin{pmatrix} 1 & 2 \\ 3 & -4 \end{pmatrix}$. Since $\det K = -7 \equiv 22 \bmod 29$, the key matrix K is invertible. Our message converts to the plaintext matrix $P \equiv \begin{pmatrix} 7 & 11 & 14 \\ 4 & 11 & 26 \end{pmatrix} \bmod 29$ using the 29-letter alphabet. Then

$$\begin{aligned} C &\equiv PK \bmod 29 \\ &\equiv \begin{pmatrix} 15 & 33 & 66 \\ 5 & -11 & -62 \end{pmatrix} \bmod 29 \\ &\equiv \begin{pmatrix} 15 & 4 & 8 \\ 5 & 18 & 25 \end{pmatrix} \bmod 29 \end{aligned}$$

is the ciphertext matrix. This corresponds to the ciphertext message `PFESIZ`.

The encryption formula did not require the key matrix to be invertible. However, decryption requires invertibility! To see this, we solve the encryption formula for P to obtain $P \equiv K^{-1}C \bmod 29$. If K is not invertible, the formula does not make sense.

18.1 Breaking the Hill cipher

Though the encryption mechanism for the Hill cipher appears more complex than for some other ciphers, it is easy to break if we have enough information. It turns out that we need to know only four letters of the plaintext message to break a message of any length!

To see this, we will look at an example.

Example 18.2. Suppose we want to crack the ciphertext message `CJUWKHN,RFI_` that was encrypted using a Hill cipher. Suppose we suspect, for whatever reason, that the message begins with the letters "chuc" but do not know the key matrix. We only know the following parts of the encryption formula, once we have converted the ciphertext message to a matrix:

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} 2 & 20 & * & * & * & * \\ 7 & 2 & * & * & * & * \end{pmatrix} \equiv \begin{pmatrix} 2 & 20 & 10 & 13 & 17 & 8 \\ 9 & 22 & 7 & 27 & 5 & 26 \end{pmatrix} \bmod 29$$

We want to find the four unknown entries a, b, c, and d of the key matrix K. Once we know them, we can find the inverse of the key matrix and use the decryption formula to obtain the entire plaintext message. We multiply to obtain the following equivalences:

$$\begin{aligned} 2a + 7b &\equiv 2 \bmod 29 \\ 2c + 7d &\equiv 9 \bmod 29 \\ 20a + 2b &\equiv 20 \bmod 29 \\ 20c + 2d &\equiv 22 \bmod 29 \end{aligned}$$

We can use these equivalences to solve for the unknowns, finding that $a = c = d = 1$ and $b = 0$. This gives the key matrix

$$K = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$$

and "Chuck Norris" as the plaintext message.

18.2 Exercises

1. The ciphertext

   `MDIVW,BZQYYVTXNREHROQZZPGKJVQZDR`

   was encrypted using the Hill cipher with key matrix $K = \begin{pmatrix} 4 & 2 \\ 42 & 42 \end{pmatrix}$. What is the message?

2. The ciphertext

   `RAXJWBX.EQ_DRJSX`

   was encrypted using the Hill cipher with key matrix $K = \begin{pmatrix} 3 & 7 \\ 12 & 19 \end{pmatrix}$. What is the message?

3. Find two 2×2 modular matrices that could not be used as key matrices for a Hill cipher (working mod 29).

4. Find two 2×2 modular matrices that could not be used as key matrices for a Hill cipher (working mod 26).

5. Suppose you receive the ciphertext

   `A,DTOJLF.DHFAZRUQUPQ`

   and suspect it is a news update encrypted using the Hill cipher. Use the plaintext crib NEWS to determine the entire message.

Chapter 19

Permutations

The theory of *permutations* deals with rearrangements. It has great applicability in several areas of mathematics, and will come in handy later in this course.

Definition 19.1. A *permutation* of a collection of objects is an ordering (or rearrangement) of those objects. In particular, a *permutation on n numbers* is a permutation of the numbers $1, 2, \ldots, n$.

We have already seen an application of this definition when discussing combinatorics. We saw that there are $n!$ ways to order n objects, so there are $n!$ permutations on n numbers. We need ways to specify permutations. One easy way is to specify a permutation using a table.

Example 19.2. The following table defines a permutation on 5 numbers:

n	1	2	3	4	5
$\pi(n)$	5	3	1	4	2

The table specifies where each number n is placed under the permutation π.

A more convenient way to specify a permutation involves *cycles*. To identify a cycle, we see where a number is placed under a permutation, then where the resulting number is placed, and so on.

Example 19.3. In our previous example, note that we can follow numbers in the following way:

$$\begin{aligned} \pi(1) &= 5 \\ \pi(5) &= 2 \\ \pi(2) &= 3 \\ \pi(3) &= 1 \end{aligned}$$

In this way, we have "cycled" back to our starting number. We notate this cycle as (1 5 2 3). We follow up with the next unused number and see that $\pi(4) = 4$, a cycle with only one number. We notate this cycle as (4). Since there are no numbers left, we would write $\pi = (1\ 5\ 2\ 3)(4)$. This is called *cycle notation*.

Given a permutation in cycle notation, it is easy to see how numbers are rearranged.

Example 19.4. Suppose π is a permutation on seven numbers and is expressed in cycle notation as $\pi = (1\ 4\ 6)(2\ 3)(5\ 7)$. Then we have $\pi(3) = 2$ and $\pi(4) = 6$.

An important property of cycle notation is that cycles can be rearranged in a looping fashion, provided the order of numbers is not affected.

Example 19.5. We have the following equalities of cycles:

$$(1\ 2\ 3\ 4) = (2\ 3\ 4\ 1) = (3\ 4\ 1\ 2) = (4\ 1\ 2\ 3)$$

By convention, we rearrange cycles to start with the lowest possible numbers. An important, but simple, permutation leaves all numbers unchanged.

Definition 19.6. The *identity permutation* on n numbers is the permutation

$$1_n := (1)(2)\cdots(n)$$

that leaves all numbers unchanged.

19.1 Permutation operations

It is common to combine two permutations. When doing so, we rearrange numbers according to the first permutation, then rearrange them again according to the second. The result is a new permutation.

Example 19.7. Suppose we have the following permutations on six numbers:

n	1	2	3	4	5	6
$\pi(n)$	4	1	6	2	3	5
$\sigma(n)$	6	2	4	5	1	3

Suppose we want a new permutation that first applies π and then σ. We would denote this permutation as $\sigma\pi$ (note this may seem backwards from what we might expect; we read the letters right to left). Note that $\pi(1) = 4$, and that $\sigma(4) = 5$. Then $\sigma\pi(1) = 5$. We end up with the following:

n	1	2	3	4	5	6
$\sigma\pi(n)$	5	6	3	2	4	1

We could write this in cycle notation as $\sigma\pi = (1\ 5\ 4\ 2\ 6)(3)$.

Definition 19.8. If π and σ are two permutations on n numbers, the *composition* $\sigma\pi$ is the permutation on n numbers that results from first applying π, and then applying σ.

Notice that this definition requires both permutations in a composition to be on the same numbers. If they are not, the operation does not make sense.

Given permutations π and σ, it is easy to find $\sigma\pi$ if the original permutations are expressed in cycle notation.

Example 19.9. From our previous example, we can write $\pi = (1\ 4\ 2)(3\ 6\ 5)$ and $\sigma = (1\ 6\ 3\ 4\ 5)(2)$ in cycle notation. By following numbers in these cycles, we find the cycle notation used above.

We are often interested in what happens when we "undo" a permutation. That is, instead of asking where a given number is placed under a permutation, we may want to know where a given number came from. While we can compute an inverse permutation using tables, an easier way is to write each cycle in the original permutation in reverse order.

Definition 19.10. Given a permutation π on n numbers, the *inverse* permutation π^{-1} is the permutation on n numbers such that $\pi\pi^{-1} = \pi^{-1}\pi = 1_n$.

Example 19.11. Let $\pi = (1\ 4\ 6\ 7)(2\ 9\ 3)(5\ 8)$ be a permutation on 9 numbers. Then

$$\pi^{-1} = (7\ 6\ 4\ 1)(3\ 9\ 2)(8\ 5) = (1\ 7\ 6\ 4)(2\ 3\ 9)(5\ 8)$$

is the inverse.

There is one last operation on permutations that will come in handy later.

Definition 19.12. Let π and σ be permutations on n numbers. The *conjugation* of π by σ is the permutation $\sigma\pi\sigma^{-1}$ on n numbers.

There are useful facts about conjugation that relate it to cycle structure.

Fact 19.13. *The cycles of $\sigma\pi\sigma^{-1}$ can be obtained from those of π by replacing each element i with $\sigma(i)$.*

Proof. Since π maps $i \mapsto \pi(i)$, it suffices to show that $\sigma\pi\sigma^{-1}$ maps $\sigma(i) \mapsto \sigma\pi(i)$. Note that we have $\sigma\pi\sigma^{-1}(\sigma(i)) = \sigma\pi 1(i) = \sigma\pi(i)$. □

Fact 19.14. *The cycle structure (that is, the number of entries per cycle) of $\sigma\pi\sigma^{-1}$ is the same as that of π, though the numbers appearing in the cycles may differ.*

Proof. This follows from the previous result, since there is a direct correspondence between the elements in the cycles of π and $\sigma\pi\sigma^{-1}$. □

19.2 Exercises

1. Let π be a permutation on 12 numbers that has the following ordering:

n	1	2	3	4	5	6	7	8	9	10	11	12
$\pi(n)$	5	8	7	1	11	2	3	6	9	12	10	4

   (a) Write π in cycle notation.

   (b) Find π^{-1} and write it in cycle notation.

2. Let $\sigma = (1\ 3\ 4\ 7)(2\ 5\ 8\ 6\ 9)$ and $\pi = (1\ 3)(2\ 9\ 7)(4\ 6\ 5)(8)$ be two permutations on 9 numbers.

   (a) Find $\sigma\pi$.

   (b) Find $\pi\sigma$.

   (c) Compute $\pi\sigma\pi^{-1}$ and see if the number of entries in each cycle matches the number of entries in each cycle of σ.

   (d) Compute $\sigma\pi\sigma^{-1}$ and see if the number of entries in each cycle matches the number of entries in each cycle of π.

3. A *derangement* on n numbers is a permutation of n numbers where no number goes to itself; that is, it is a permutation with no single cycles.

   (a) Determine the number of derangements on one number.

   (b) Determine the number of derangements on two numbers.

   (c) Determine the number of derangements on three numbers.

   (d) Determine the number of derangements on four numbers.

   (e) Let D_n denote the number of derangements on n numbers. Use the following method to show that $D_n = (n-1)(D_{n-1} + D_{n-2})$. We can view the problem as the number of ways to distribute n hats to n people so no one gets her own hat. Suppose the first person takes hat i. Think of how many ways this can be done. If person i takes the first hat, the problem reduces to one with how many people? If person i does not take the first hat, then each of the $n-1$ people (not counting the first person, who already picked a hat) has only one hat they cannot choose. How does this reduce to a problem with fewer people?

4. Show the following properties of permutations on n numbers.

   (a) The composition of two permutations is a permutation.

   (b) There is a permutation that leaves every permutation unchanged under composition.

   (c) Given any permutation, there is a permutation that undoes it.

5. Let $\sigma = (1\ 3\ 8\ 9)(2\ 4\ 6\ 5)(7)$ and $\pi = (1)(2\ 9\ 4)(3\ 5\ 7)(6\ 8)$. For this exercise, write all permutations in cycle notation.

(a) Compute the inverse π^{-1}.

(b) Compute the composition $\pi\sigma$.

(c) Compute the conjugation $\pi\sigma\pi^{-1}$.

(d) Compute σ^{-1}.

(e) Compute the composition $\sigma\pi$.

(f) Compute the conjugation $\sigma\pi\sigma^{-1}$.

(g) Do you notice any pattern between the original permutations and their conjugations?

6. Set an Enigma to day key 213 SKL PL-1. For this exercise, write all permutations in cycle notation.

(a) Compute the permutation P of the alphabet that results from the plugboard.

(b) Explain why $P = P^{-1}$.

(c) Compute the permutation R of the alphabet that results from the sequence of rotors, reflector, and rotors; that is, the Enigma encryption process without the plugboard.

(d) Compute the composition PRP^{-1}. Notice the cycle structure of PRP^{-1} (encryption with the plugboard) matches that of R (encryption without the plugboard).

Chapter 20

Rejewski attack on Enigma

A successful crack of the Enigma cipher was discovered by Marian Rejewski, a Polish mathematician. His method uses permutation theory to determine both the plugboard and rotor settings. We outline his method with an extended example.

Suppose we want to decrypt the intercepted Enigma ciphertext `HFJSOYYBLATUO` but do not know the machine settings. Rejewski's attack uses other intercepted ciphertext messages. Suppose that on the same day, we intercepted many other messages, the first six letters of which are the following:

```
WKOTFI BOLJRV JOSURM EFKBOT RBEDAP TBHCAX HWKSBT YQDZNS EBXBAB KZXAQB DABNUW QFMQOF
WEOTSI UWGMBN WRBTJW WLDTVS ZYDKMS FAREUC XXHXKX DGDNXS NNSHDM QKXQFB CCZFLH VCHVLX
ADPRWQ XQUXNA JHJUGY TULCYV PFYWOL NQVHNG YKIZFK GGDGXS BSXJEB TITCTZ SZALQR KKDAFS
SSVLEG IICITU LPSYZM OGKOXT LXRYKC MOXPRB SLNLVE KTFAID XVAXHR HFJSOY JJQUCJ DMWNPO
REJDSY XUZXYH
```

While each of these messages used the same day key, each has a different unknown message key. In fact, the first six letters of each intercepted message must be the encrypted three-letter message key repeated. But what are we to do since we don't know any of the message keys?

Consider the encrypted message key `WKOTFI` from our list. We do not know what the plaintext is, but we do know it consists of three letters repeated. Let's suppose (almost certainly incorrectly) that "abcabc" is the corresponding plaintext. Since the Enigma cipher is polyalphabetic, each rotor position gives some permutation of the alphabet. Therefore, the first six ciphertext letters in each message are the mappings of some letters under the first six Enigma permutations. Call these permutations $P_1, \cdots, P_6$. We then guess, using our plaintext assumption, the following mappings:

$$\begin{aligned} P_1 : \text{a} &\leftrightarrow \texttt{W} \\ P_2 : \text{b} &\leftrightarrow \texttt{K} \\ P_3 : \text{c} &\leftrightarrow \texttt{O} \\ P_4 : \text{a} &\leftrightarrow \texttt{T} \\ P_5 : \text{b} &\leftrightarrow \texttt{F} \\ P_6 : \text{c} &\leftrightarrow \texttt{I} \end{aligned}$$

Notice that these mappings work both ways, since the Enigma cipher is symmetric at each permutation. While we know very little about each permutation (remember, our plaintext guess is almost certainly wrong), we can say more about compositions of these permutations. Since P_1 maps $\texttt{W} \mapsto$ a and P_4 maps $\text{a} \mapsto \texttt{T}$, the composition P_4P_1 maps $\texttt{W} \mapsto \texttt{T}$. This means if we consider the composition, our guess does not matter! Similarly, we know the composition P_5P_2 maps $\texttt{K} \mapsto \texttt{F}$ and the composition P_6P_3 maps $\texttt{O} \mapsto \texttt{I}$. Continuing in this fashion, we see that each encrypted message key in our intercepted list gives a mapping for each of three compositions. Our goal is to find where each letter maps under these compositions. The process is tedious but straightforward.

	P_4P_1	P_5P_2	P_6P_3
A	R	U	R
B	J	A	W
C	F	L	U
D	N	W	S
E	B	S	P
F	E	O	D
G	G	X	N
H	S	G	X
I	I	T	K
J	U	C	Y
K	A	F	T
L	Y	V	V
M	P	P	F
N	H	D	E
O	O	R	I
P	W	Z	Q
Q	Q	N	J
R	D	J	C
S	L	E	M
T	C	I	Z
U	M	Y	A
V	V	H	G
W	T	B	O
X	X	K	B
Y	Z	M	L
Z	K	Q	H

The next step in Rejewski's attack is to write each of the three compositions in cycle notation. The reason for this will become apparent shortly.

$$\begin{aligned} P_4P_1 &= (\text{ARDNHSLYZK})(\text{BJUMPWTCFE})(\text{G})(\text{I})(\text{O})(\text{Q})(\text{V})(\text{X}) \\ P_5P_2 &= (\text{AUYMPZQNDWB})(\text{CLVHGXKFORJ})(\text{ES})(\text{IT}) \\ P_6P_3 &= (\text{ARCU})(\text{BWOIKTZHX})(\text{DSMF})(\text{EPQJYLVGN}) \end{aligned}$$

The next step is to count, for each of the three compositions, the number of letters in each cycle. This collection is numbers is called the *signature* for a given Enigma setting. By convention, we write these numbers in descending order. The signature for our example is $(10-10-1-1-1-1-1-1, 11-11-2-2, 9-9-4-4)$.

Before we proceed further, we need to understand exactly how the Enigma machine produces these permutations. Think about the path that electrical signal takes in the machine when a key is pressed for some rotor configuration. The signal passes through the plugboard, the rotors, the reflector, the rotors again, and finally the plugboard again. Notice that the plugboard acts as a permutation; call it P. Further, the rotor-reflector-rotor path acts as a complicated permutation as well; call it R_i since the rotors move frequently. Then we may write the permutation $P_i = PR_iP$. Since the plugboard only swaps letters, we have $P = P^{-1}$, so $P_i = PR_iP^{-1}$.

This construction should look familiar. The previous equation tells us that P_i is the conjugation of R_i by P. We have already seen that the cycle structure of P_i and R_i must be the same. This is the reason for computing the cycle structure of our compositions!

There is one problem: this analysis only applies to a given permutation on the Enigma machine. We are dealing with compositions. Notice, however, that for Enigma permutations P_i and P_j we have $P_iP_j = (PR_iP^{-1})(PR_jP^{-1}) = PR_iR_jP^{-1}$. The same analysis holds!

Rejewski and his team realized that the cycle structure of permutation compositions with the plugboard (like P_4P_1) must be the same as the cycle structure of permutation compositions without the plugboard (like R_4R_1). Why was this important? If they were to compute every possible cycle structure with the plugboard,

their effort would have taken an unimaginable amount of time. But by ignoring the plugboard, there are only about a hundred thousand settings for which they needed the cycle structure. The team spent a year doing so, and catalogued the results in a series of books; perhaps we will call these books an "Enictionary" for brevity.

Now that we have already computed the signature for our message, we would look it up in the Enictionary. The Enictionary entry would list the signature $(10-10-1-1-1-1-1-1, 11-11-2-2, 9-9-4-4)$ along with the rotor order and orientation that yields that signature. In this case, it gives three entries: `123 NMZ`, `213 XRE`, and `231 ZQP`. This is still a bit ambiguous, but far better than having to check one setting for every possible plugboard setting that could have been used. For the sake of brevity, we will simply state that the settings for our example are `231 ZQP`. In reality, we would need to perform the subsequent analysis on each of the three possibilities.

Now that we know the rotor order and orientation, we will recover the plugboard settings used in our message. To do this, we will need to determine where each letter is mapped under the permutations R_4R_1, R_5R_2, and R_6R_3. Remember, these are the permutation compositions with no plugboard.

	R_1	R_2	R_3	R_4	R_5	R_6
A	M	V	Q	G	L	J
B	C	O	N	I	C	F
C	B	W	M	J	B	E
D	V	F	H	V	S	V
E	N	Z	Z	L	U	C
F	I	D	X	N	T	B
G	Y	L	L	A	P	X
H	W	Q	D	T	N	K
I	F	J	U	B	R	Z
J	Z	I	T	C	W	A
K	U	U	Y	M	Y	H
L	R	G	G	E	A	Q
M	A	P	C	K	X	R
N	E	R	B	F	H	W
O	X	B	V	X	Z	P
P	Q	M	W	Q	G	O
Q	P	H	A	P	V	L
R	L	N	S	S	I	M
S	T	T	R	R	D	U
T	S	S	J	H	F	Y
U	K	K	I	Z	E	S
V	D	A	O	D	Q	D
W	H	C	P	Y	J	N
X	O	Y	F	O	M	G
Y	G	X	K	W	K	T
Z	J	E	E	U	O	I

From this we find the cycle structure of each of the three compositions:

$$\begin{aligned} R_4R_1 &= (\text{AKZCINLSHY})(\text{BJUMGWTREF})(\text{D})(\text{O})(\text{P})(\text{Q})(\text{V})(\text{X}) \\ R_5R_2 &= (\text{AQNIWBZUYMG})(\text{CJRHVLPXKEO})(\text{DT})(\text{FS}) \\ R_6R_3 &= (\text{ALXBWODKT})(\text{CRUZ})(\text{EISM})(\text{FGQJYHVPN}) \end{aligned}$$

The final step in determining the plugboard settings is to compare the cycles of, for example, P_4P_1 with those of R_4R_1. Based on our knowledge about the relationship of elements in the cycles of a permutation and its conjugate, if we rearrange the cycles, we should be able to identify which letters are swapped by the plugboard. Doing so, we obtain the following:

$$\begin{aligned} R_4R_1 &= (\text{BJUMGWTREF})(\text{CINLSHYAKZ})(\text{P})(\text{D})(\text{O})(\text{Q})(\text{V})(\text{X}) \\ P_4P_1 &= (\text{BJUMPWTCFE})(\text{RDNHSLYZKA})(\text{G})(\text{I})(\text{O})(\text{Q})(\text{V})(\text{X}) \end{aligned}$$

This gives the plugboard setting G/P,I/D,A/Z,E/F,C/R,H/L. The message decrypts to "catcatawesome" in plaintext.

20.1 Exercises

1. Suppose the Enigma ciphertext `PCOXLRUOQCULRSD` was intercepted on a certain day. Further, the following additional encrypted message keys were intercepted on the same day:

   ```
   QGEQIB EMKDXH QLUQFG RQETYB NOJLQJ EHTDAA PCFXLW FWJSDJ OARVJO HUTEWA ZONZQZ RSGTHQ
   PAVXJS DIGIPQ XPLOKX YGCYIF VVJJVJ BSVFHS SXOHTR KHDWAC SPTHKA LWTMDA AWTADA JYQKCU
   NACLJF XKEOSB WECGUF BNIFGE JRCKRF IXCRTF PTYXNY VOUJQG LIKMPH MZAPEM KRIWRE QJHQMK
   IOXRQL GZVNES JXTKTA LBMMZT EFPDBD UDZCON KELWUX OFUVBG TVSUVI BPBFKV CTWBNP BCAFLM
   ```

   Determine the signature and ask for the corresponding rotor settings. Then determine the plugboard settings and decrypt the message.

2. Suppose the Enigma ciphertext `SRNSJZWCMSPUDRNRGUCLOG` was intercepted on a certain day. Further, the following additional encrypted message keys were intercepted on the same day:

   ```
   YPBKRS MDAJYV WJNWVZ LBKRXH SFTSGU NEPNZX RHVZEK ZGSPLD PQLTMR UCZBTO XODAOG DMIXHP
   BLXUIN FSCFPM QKWQKJ IAFHNL OIROUT GTQDAI ENHEWA VREIJQ JWYMQE TZGYCB CUOCSY KYMLDF
   AXUVBW HVJGFC
   ```

   Determine the signature and ask for the corresponding rotor settings. Then determine the plugboard settings and decrypt the message.

Chapter 21

Number systems

How do we write a number? This question seems silly, but it turns out that a formal understanding of the digits of numbers can be extended in useful ways, particularly when studying how computers handle computations. Formally, we write decimal numbers using powers of ten, though we don't often think about them in this way.

Example 21.1. When we write the integer 2031, we really mean $2031 = 2 \times 10^3 + 0 \times 10^2 + 3 \times 10^1 + 1 \times 10^0$.

This notation justifies our use of placeholders like the "hundreds place" when learning how to write numbers. When we expand decimal numbers as sums of powers of ten, the digits multiplying each power never exceed nine. If they did, we would simply increase the next highest power instead. This is how we add decimal numbers, in fact, when we "carry over" numbers to higher placeholders.

There is nothing special about writing numbers using powers of ten, except that humans have ten fingers and ten toes. We could easily (well, once we learn how it works) write numbers using powers of a different number instead. These methods are called *number systems*, and there are several common ones that we will consider. Fortunately, the method we will use is quite general.

Example 21.2. We could write

$$\begin{aligned} 2031 &= 2 \times 10^3 + 0 \times 10^2 + 3 \times 10^1 + 1 \times 10^0 \\ &= 3 \times 8^3 + 7 \times 8^2 + 5 \times 8^1 + 7 \times 8^0 \end{aligned}$$

instead. We might write this representation, using sums of powers of eight, as $2031_{10} = 3757_8$ to indicate what number we use when taking powers.

In general, we can express a number using sums of powers of any positive integer we like. In fact, given a choice of what number we use for taking powers, each integer can be written uniquely in this way.

Fact 21.3. *Let a be a positive integer. Then for any integer $b > 1$ there is a unique representation $a = a_n b^n + a_{n-1} b^{n-1} + \cdots + a_1 b^1 + a_0 b^0$ of a, where $0 \leq a_i < b$ for each i. We write $a = (a_n a_{n-1} \ldots a_1 a_0)_b$ to denote this (and often drop the parentheses when there is no confusion).*

We typically call b the *base* of the number system. When $b = 10$, we call the system the *decimal* system and call each a_i a *digit* or *decimal digit*. The most common number systems other than decimal are *binary* ($b = 2$), *octal* ($b = 8$), and *hexadecimal* ($b = 16$). In the case of binary, we call each a_i a *bit* instead of a binary digit.

In binary, the acceptable bits are numbers in the set $\{0, 1\}$. In octal, they are numbers in the set $\{0, 1, 2, 3, 4, 5, 6, 7\}$. But what about hexadecimal? We are allowed digits up to 15, which could cause confusion when we write out numbers. To solve this, we use digits in the set $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, \mathrm{A}, \mathrm{B}, \mathrm{C}, \mathrm{D}, \mathrm{E}, \mathrm{F}\}$ instead to reduce confusion. In the next section, we will see how this works in more detail.

21.1 Converting general bases to decimal

The conversion of a number written in a general base to decimal is straightforward and seen easily with examples.

Example 21.4. To convert the integer $2E7D_{16}$ to decimal, we write

$$\begin{aligned} 2E7D_{16} &= 2 \times 16^3 + 14 \times 16^2 + 7 \times 16^1 + 13 \times 16^0 \\ &= 8192_{10} + 3584_{10} + 112_{10} + 13_{10} \\ &= 11901_{10}. \end{aligned}$$

When performing these steps, we may drop subscripts in the intermediate steps when using decimal numbers since we use them so often.

Example 21.5. To convert the integer 100101_2 to decimal, we write

$$\begin{aligned} 100101_2 &= 1 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \\ &= 32 + 4 + 1 \\ &= 37_{10}. \end{aligned}$$

It is in general not as easy to convert numbers from decimal to a general base. However, there are several methods that we can use for specific bases, namely binary and hexadecimal.

21.2 Converting decimal to binary

We will examine two algorithms for converting decimal numbers to their binary equivalents. The first operates largely by inspection and is useful for small numbers.

Example 21.6. Suppose we want to convert the decimal number 123_{10} to binary. We need to know the largest power of two that is less than 123; by inspection, it is $64 = 2^6$. We then subtract to write $123 = 1 \times 2^6 + 59$. We then need the largest power of two that is less than 59; it is $32 = 2^5$. We subtract again to write $123 = 1 \times 2^6 + 1 \times 2^5 + 27$.

Continuing in this fashion, we obtain the representation $123_{10} = 1111011_2$.

This method is not very efficient, and requires us to know or compute powers of two. There is a much faster way that works efficiently for numbers of any size. The algorithm works using these steps:

1. Write the decimal number.
2. If the number is even, mark a zero as its last bit and divide the number by two.
3. If instead the number is odd, mark a one as its last bit, subtract one from the number and divide it by two.
4. If this new number is even, mark a zero as the second-to-last bit and divide the number by two.
5. If instead the new number is odd, mark a one as the second-to-last bit, subtract one from the number and divide it by two.
6. Continue this process, determining each successive bit right-to-left.
7. When the number becomes zero, stop.

Example 21.7. We will verify that $123_{10} = 1111011_2$ using this algorithm.

123	
61	1
30	11
15	011
7	1011
3	11011
1	111011
0	1111011

21.3 Converting decimal to hexadecimal

We may occasionally wish to convert a decimal number to hexadecimal, though this is less common. Before we state the algorithm, notice that a four-bit binary integer always takes on a value between zero (0000_2) and fifteen (1111_2). It is no coincidence that these are precisely the values that a single hexadecimal digit may assume. It is this fact that permits the following algorithm:

1. Convert the decimal number to binary.
2. Working right-to-left, mark the bits in groups of four. If necessary, add zeros to the front of the number to form the last group.
3. Replace each group of four bits by the corresponding hexadecimal digit.

Example 21.8. We want to find the hexadecimal value of 123_{10}. We have already found its binary value as 1111011_2. We group the bits, adding a single zero to the front, to obtain 0111 1011. The group 0111_2 has hexadecimal value 7_{16} and the second has value B_{16}, so $123_{10} = 7\mathrm{B}_{16}$.

21.4 XOR

The exclusive OR (XOR) operation takes two binary numbers as its inputs and outputs a single binary number. The XOR operation is used in many cryptographic algorithms. The operation uses the $\oplus$ symbol and operates on two binary integers X and Y using these steps:

1. If the two numbers do not have the same number of bits, add enough zeros to the front of the shorter one to make them the same length. This does not change the value of the number.
2. Compare the leftmost bits. If they are the same, the leftmost bit of $X \oplus Y$ is a zero. If they are different, it is a one.
3. Compare the next leftmost bits and perform the same check.
4. Continue for all bits.

Example 21.9. Suppose we want to find the value of $110100101 \oplus 011010010$.

$$\begin{array}{rr} & 110100101 \\ \oplus & 011010010 \\ \hline & 101110111 \end{array}$$

There are important properties of the XOR operation that are presented in the exercises.

21.5 Exercises

1. Write your birth year in binary. Be sure to show all steps.
2. Write the decimal number 31415 in binary. Verify your answer is correct by converting it back to decimal.
3. Write the hexadecimal numbers DEAD_{16} and BEEF_{16} in binary and in decimal.
4. Show that $X \oplus X = 0$ for any X.
5. Show that $(X \oplus Y) \oplus Z = X \oplus (Y \oplus Z)$.
6. Show that $X \oplus Y = Y \oplus X$ for any X and Y.
7. Suppose that $P \oplus K = C$. Explain why $C \oplus K = P$ and show an example.
8. Find a way to use the XOR operation to determine if the number of one bits in a binary number is odd.
9. Explain how to count to 1023 using just your fingers.

Chapter 22

Randomness and stream ciphers

Almost all of modern cryptography relies on random numbers. Even a classical cryptographic system like the one-time pad is most secure when the key is a random set of letters. How can we generate a long set of random numbers? Since we can write any number using any base, an equivalent problem is generating a long random string of zeros and ones. Several options are used:

- Mathematical algorithms
- Computer events
- Random physical phenomena
- Electrical properties

It is important to distinguish which of these techniques generate "actually" random numbers or "mostly" random numbers. There are technical definitions of true randomness, but it suffices for us to know that mathematical algorithms are *pseudorandom* number generators (PRNGs) and computer events, random physical phenomena, and electrical properties can be used as true random number generators (RNGs). Electrical RNGs often use a series of diodes, while computer event RNGs use things like keyboard input, microphone noise, and file operations. Physical RNGs use random phenomena like nuclear decay or cosmic ray activity. A simple RNG could be repeated coin flips, using heads for a one bit and tails for a zero bit.

In contrast, mathematical algorithms operate as PNGs, since they provide a method for generating a string of bits given a starting value, the *seed*. An important property of these methods is that the same seed will always produce the same string of bits. Therefore, the seed should be a truly random number. Since PRNGs are typically much faster than RNGs, it is common to use the output from an RNG to seed a PRNG and periodically start the PRNG with a new random seed.

22.1 Linear feedback shift register

We will use our knowledge of binary numbers and operations to construct a simple PRNG, the *linear feedback shift register* (LFSR). An LFSR takes as its input two pieces of information: a binary number seed and a rule for generating new bits from old ones. Starting with the seed, an LFSR uses its rule to generate a single new bit, then uses this new bit and a portion of the original seed to generate another new bit. It then continues as long as numbers are needed. The bits output by an LFSR are often called a *bit stream* or simply a *stream*. For this reason, an LFSR is called a *stream cipher*.

Typically, an LFSR rule will apply the XOR operation to some of the last bits it has generated. We examine an example to see how rules are typically constructed.

Example 22.1. We wish to generate pseudorandom numbers using an LFSR. We flip a coin five times to obtain the five-bit seed 11010_2. For the rule, we will XOR the last bit, third-from-last bit, and fourth-from-last bit. Since an LFSR always starts with its seed, we already know the beginning of the stream:

11010

The first application of the rule performs the operation $0 \oplus 0 \oplus 1 = 1$, so the next bit in the stream is a one. This yields the following stream:

$$110101\underline{1}$$

Notice that for the next step, the bits used have changed. Now we perform the operation $1 \oplus 1 \oplus 0 = 0$:

$$1101010\underline{0}$$

We could continue this process, generating as many bits as we like.

We might ask why we don't use all the previously generated bits in some way. Since LFSRs might be implemented in hardware, it might be infeasible to store all the bits generated. If we use a rule-based LFSR, we only need to keep track of a few bits at a time.

Fact 22.2. *Any linear feedback shift register eventually repeats its bit stream.*

The proof of this fact is an exercise.

It is possible to use the bit stream from an LFSR in a cryptographic system. A common way to encrypt a message using an LFSR is the following algorithm:

1. Convert a plaintext message to binary (explained below).
2. Use an LFSR to generate as many bits as are in the binary plaintext message. The bit stream is the key.
3. Apply the XOR operation to each plaintext bit and corresponding key bit.
4. The resulting bits from the XOR operation form the binary ciphertext message.
5. If desired, convert the ciphertext message to letters (again, explained below).

How would we decrypt a message encrypted in this way?

In order to use an LFSR for this algorithm, we need a way to represent messages in binary. There are many schemes available for doing so and, in fact, this is how computers represent letters internally. Some common schemes are ASCII and Unicode. These schemes provide binary encodings for many symbols besides the alphabet, so we will generate our own simpler scheme. We want our scheme to encode the entire 29-character alphabet (to include a space, comma, and period). This will require five bits. The simplest way to do so is to encode each letter as the binary representation of its decimal equivalent. We include the full encoding scheme in Appendix G for easy reference.

Example 22.3. Suppose we want to encrypt the message "sunday" using an LFSR with five-bit seed 11010_2 and rule of XORing the second-, third-, and fifth-from-last bits. We convert the message to binary using the scheme in Appendix G:

$$10010101000110100011000001100 0$$

To apply the LFSR cipher, we need thirty key bits. The LFSR generates the following bit stream:

$$11010000101111010000101111010 0$$

To find each ciphertext bit, we XOR each plaintext bit with its corresponding key bit. This yields the following ciphertext bits:

$$01000101101001110011101110110 0$$

To find the letter representation of the ciphertext message, break the ciphertext into five-bit groups:

$$01000101101001110011101110110 0$$

The corresponding ciphertext message is `IWTTXM`.

22.2 RC4

One of the most well-known stream ciphers was developed by cryptographer Ronald Rivest (of RSA fame) in 1987. While its operation is very different from a linear feedback shift register, it is a stream cipher and can be used for encryption and decryption by a simple XOR operation. Like any stream cipher, it generates a stream of pseudorandom bits that are combined with message bits.

It is intuitive to study RC4 by first examining a smaller variant of it (made up for this purpose) that we will call *BabyRC4*. The procedures and algorithms behind BabyRC4 are identical to those of actual RC4 in every way, although the bit lengths have been reduced to be more manageable to do by hand.

22.2.1 State preparation

The first step in the BabyRC4 algorithm is to use an initial key to prepare an array of bits for later use. We begin by setting up a permutation of 8 numbers, called S (for state), in order: $S = 01234567$ The key is an array of numbers K, each between 0 and 8, that is up to 8 numbers long. Suppose we use the key $K = 34812$. There are two counters, i and j, each initially set to zero.

For each value of i from 0 to 7, we perform the following steps:

1. Compute a new value $j = (j + S(i) + K(i \bmod 5)) \bmod 8$. That is, we use the current counter j, one of the state values, and one of the key values (looping around as needed) to set a new counter j.

2. Swap the state entries in the i and j positions. That is, swap $S(i)$ and $S(j)$.

We show the process in order for the key we chose.

i	j	S (after swap)
0	0	01234567
0	3	31204567
1	0	13204567
2	2	13204567
3	3	13204567
4	1	14203567
5	1	15203467
6	3	15263407
7	2	15763402

Notice that even with these few steps, the initial state permutation is well mixed.

22.2.2 Stream generation

We are now ready to begin generating stream bits to XOR with a message for encryption. We begin by setting the counters i and j to zero again. To generate a stream bit, we perform the following steps:

1. Increment $i = (i + 1) \bmod 8$, looping around as needed.

2. Compute a new value $j = (j + S(i)) \bmod 8$. That is, we use the current counter j and one of the state values to set a new counter j.

3. Swap the state entries in the i and j positions. That is, swap $S(i)$ and $S(j)$.

4. Output the state bit at the $(S(i) + S(j)) \bmod 8$ position. That is, the entries in the i and j positions are used to specify another position that contains the output bit.

Note that the stream generation steps are very similar to the key scheduling process, where we use counters to continually modify the state permutation, helping ensure good pseudorandomness.

We show the process in order for the key we chose. In theory, we could continue the process indefinitely, but we will show only the first ten output values.

i	j	S (after swap)	output
0	0	15763402	-
1	5	14763502	4
2	4	14367502	3
3	2	14637502	4
4	1	17634502	3
5	6	17634052	0
6	3	17654032	1
7	5	17654230	6
0	6	37654210	4
1	5	32654710	2
2	3	32564710	6

You might notice that in order to XOR the stream with a message, both need to be expressed in bits, which this stream is not. However, notice that the output stream is numbers from 0 to 8, which are precisely the three-bit numbers. Hence, each output number actually provides three bits (when we pad with zeros as needed) to XOR with a message.

22.2.3 Quality of output

The goal of a good stream cipher is to provide output that is as random as possible, since any bias could help an attacker. But how good is the output of BabyRC4? To test it, we used the given key and generated 100000 output numbers, and counted the distribution of the numbers 0 through 8 that appeared. In a good stream cipher, each number would appear approximately the same number of times in such a large quantity of output. Below is a listing of the distribution.

Number	Frequency (%)
0	12.78
1	12.37
2	12.59
3	12.57
4	12.54
5	12.29
6	12.73
7	12.12

The results are excellent! Each number appears with essentially the same frequency. Of course, it should be noted that equal frequency is not a guarantee of a good stream cipher; after all, if a stream cipher repeated all the numbers in order repeatedly, it would pass the frequency test but still be entirely predictable! However, it is a good sign that BabyRC4 passes this test.

22.2.4 Extending BabyRC4 to RC4

We have so far only discussed BabyRC4. However, the extension to RC4 is trivial! In RC4, the state permutation initially contains the numbers 0 to 255 (that is, all 8-bit numbers), and the key can be up to 256 bytes (not bits!) in length. The rest of the steps proceed as in BabyRC4, accounting for the increase from working mod 8 to working mod 256.

In BabyRC4, working with a permutation on 8 numbers meant each output value was a three-bit number. In RC4, working with a permutation on 256 numbers means each output value is an eight-bit number.

22.2.5 Attacks

While RC4 is an exceptionally simple and fast stream cipher to understand and implement computationally, it has significant weaknesses. In particular, it has been shown by several groups of researchers that the output of RC4, while distributed very evenly over large outputs, contains biases in the first several bytes of output.

Further, large quantities of output (on the order of a gigabyte) have been shown to be distinguishable from truly random output by different statistical tests.

The RC4 cipher was used for wireless network security in the WEP protocol. The use of statistical biases in the output was exploited in 2001 to show that with enough encrypted output, it is possible to recover an RC4 key. This attack can be avoided by discarding some initial output bytes before use (initially recommended to be 768 bytes, but many recommend at least 3072 bytes).

Many more attacks against various implementations of RC4 have been discovered since the 2001 attack, with the result that RC4 is no longer considered safe for most purposes.

22.3 Exercises

1. Use the standard binary representation and a linear feedback shift register to encrypt the plaintext message SECRET using the five-bit seed 11110_2 and the last and fourth-from-last bits to generate the bit stream.

2. Suppose the following bit stream was generated from an LFSR using a five-bit seed and unknown XOR rule:

   $$01100101110010111010111001011$$

   Determine the seed and rule used.

3. Using any three-bit seed and XOR rule, show that the resulting LFSR bit stream repeats.

4. Explain why any LFSR must eventually repeat its bit stream.

5. If an LFSR has an n-bit seed, what is the longest bit stream that could be obtained before repeating?

6. Explain why a seed of all zeros is a terrible choice for an LFSR regardless of the XOR rule.

7. Suppose that Alice sends her bank an order to transfer money to Bob's account and uses an LFSR to encrypt the amount of the transfer. Bob does not know the seed or rule that Alice used, but he can intercept her order, change it, and send it on to the bank. Alice writes the amount of the order in binary before applying the LFSR. For example, an amount of \$100 would be written as `1100100` in binary.

   (a) Show that if Bob changes any ciphertext bits, the resulting plaintext message must be different.

   (b) Suppose that Bob changes the last bit of the ciphertext message. How might this change the resulting amount after the bank decrypts the message Bob sends along?

   (c) Suppose that Bob changes the fourth-to-last bit of the ciphertext message. How might this change the resulting amount after the bank decrypts the message Bob sends along?

   (d) Suppose that Bob changes the i^{th}-from-last bit of the ciphertext message. How might this change the resulting amount after the bank decrypts the message Bob sends along?

   (e) Explain why an LFSR does not guarantee message integrity.

   (f) Explain why this attack on an LFSR does not compromise the privacy of the message.

8. Suppose you want to use BabyRC4 for stream cipher encryption. You choose the key 123456.

   (a) Determine the state permutation S at the end of the key scheduling process.

   (b) Determine the first ten stream output values.

   (c) With only these ten output values, how many message bits could be encrypted before you need to generate additional stream output?

Chapter 23

Diffie-Hellman key exchange

In all the ciphers we have seen so far, the same key was used for encryption and decryption. This is a fundamental weakness in so-called *symmetric* ciphers: the sender must have some way to transmit the key to the recipient. This is not a trivial problem. In 1976, cryptographers Whitfield Diffie, Martin Hellman, and Ralph Merkle revolutionized modern cryptography by inventing a way for two parties to generate a key together over an insecure channel, even if they have never met. The scheme is known as *Diffie-Hellman key exchange*, unfortunately leaving out Merkle's name. This solves the aforementioned problem and is the basis for so-called *public-key cryptographic systems* that will be addressed later.

Suppose Alice and Bob wish to generate a key to use in their communications, but have never met; Alice lives in Alabama and Bob in Belgium. Unfortunately, their communication lines are being monitored by Eve. She cannot interfere with their communication, but knows everything that passes between Alice and Bob. Here is the procedure they use to implement Diffie-Hellman key exchange:

1. Alice and Bob choose a prime base g (typically a small prime like 2, 3, or 5 that should have large prime order) and very large prime modulus p. Note that Eve, who is monitoring their communication, knows g and p as well.

2. Alice chooses a secret large random number a and transmits $A = g^a \bmod p$ to Bob. Note that Eve knows A, but neither she nor Bob knows a.

3. Bob chooses a secret large random number b and transmits $B = g^b \bmod p$ to Alice. Note that Eve knows B, but neither she nor Alice knows b.

4. Alice computes $B^a \bmod p$.

5. Bob computes $A^b \bmod p$.

6. This common number is the shared key.

Why do Alice and Bob obtain the same number in this exchange? Notice that Alice computes $B^a \bmod p = (g^b)^a \bmod p = g^{ba} \bmod p$ and Bob computes $A^b \bmod p = (g^a)^b \bmod p = g^{ab} \bmod p$. The numbers $g^{ba} \bmod p$ and $g^{ab} \bmod p$ are equal since order does not matter in multiplication.

Since Eve has intercepted the numbers that Alice and Bob sent to each other, why can she not also determine the shared key? Eve knows the values of g, p, $g^a \bmod p$, and $g^b \bmod p$. However, she knows neither a nor b. There is no known operation that Eve can perform on these numbers to obtain the shared key $g^{ab} \bmod p$. One way that Eve could determine the shared key is to determine either a or b, and then perform an exponentiation to find the key. However, the problem of finding a from the quantity $g^a \bmod p$ is an extremely difficult problem known as the *discrete logarithm problem*. There is no known method to efficiently solve this problem.

23.1 Modular exponentiation

One problem we have not addressed is how to efficiently compute the exponentiations used in Diffie-Hellman key exchange. Since a, b, and p are very large numbers, the values of A and B would be infeasible to compute directly. We need a way to efficiently compute modular exponents. Fortunately, such a method exists. It is called the *square-and-multiply method* since the only computations needed are simple multiplications. To apply the square-and-multiply algorithm to compute the value of $b^x \bmod n$, first express x as a binary number. Then perform these steps:

1. Start with the leftmost bit of x. Compute $b^2 \bmod n$ and move one bit to the right in the binary expansion of n.

2. If the current bit is one, multiply the current result by b and reduce $\bmod n$.

3. If there are more bits, square the current result and move one bit to the right.

4. Repeat until there are no more bits. The result is $b^x \bmod n$.

Example 23.1. Let's demonstrate the algorithm by computing the value of $17^{20} \bmod 26$. First write the exponent as $20_{10} = 10100_2$.

Bit	Operation	Computation
1	square	$17^2 = 289 \equiv 3 \bmod 26$
0	square	$3^2 = 9 \equiv 9 \bmod 26$
1	multiply	$9 \times 17 = 153 \equiv 23 \bmod 26$
	square	$23^2 = 529 \equiv 9 \bmod 26$
0	square	$9^2 = 81 \equiv 3 \bmod 26$
0	-	-

Hence $17^{20} \equiv 3 \bmod 26$.

Example 23.2. Let's now compute the value of $23^{27} \bmod 34$. First write the exponent as $27_{10} = 11011_2$.

Bit	Operation	Computation
1	square	$23^2 = 529 \equiv 19 \bmod 34$
1	multiply	$19 \times 23 = 437 \equiv 29 \bmod 34$
	square	$29^2 = 841 \equiv 25 \bmod 34$
0	square	$25^2 = 625 \equiv 13 \bmod 34$
1	multiply	$13 \times 23 = 299 \equiv 27 \bmod 34$
	square	$27^2 = 729 \equiv 15 \bmod 34$
1	multiply	$15 \times 23 = 345 \equiv 5 \bmod 34$

Hence $23^{27} \equiv 5 \bmod 34$.

There are a few useful things to keep in mind.

- The first operation of a square-and-multiply procedure is always to square the base, even though the first bit is always a one.

- If the binary expansion of the exponent ends in a zero, no operation is performed on that bit.

- If the binary expansion of the exponent ends in a one, the only operation performed on that bit is the multiplication step.

23.2 Exercises

1. Compute $2^{20} \bmod 17$.
2. Compute $7^{30} \bmod 5$.
3. Compute $43^{59} \bmod 61$.
4. Suppose you and a friend want to use Diffie-Hellman key exchange to create a common secret. You choose the modulus $p = 101$ and the public base $g = 17$. You choose the secret exponent $a = 12$ and your friend chooses the secret exponent $b = 77$. Show that you obtain the same common secret after the key exchange.
5. Suppose you want to perform a discrete logarithm brute-force attack on a Diffie-Hellman key exchange. You know that Alice and Bob chose the modulus $p = 11$ and the public base $g = 2$. Alice transmits the number $5 \bmod 11$ to Bob, while Bob transmits the number $9 \bmod 11$ to Alice.
   (a) Find a number x such that either $2^x \equiv 5 \bmod 11$ or $2^x \equiv 9 \bmod 11$.
   (b) If $2^x \equiv 5 \bmod 11$, the shared secret is $9^x \bmod 11$. If instead $2^x \equiv 9 \bmod 11$, the shared secret is $5^x \bmod 11$.
6. Describe a way you could use Diffie-Hellman key exchange in conjunction with a linear feedback shift register to generate a keystream for use in encrypting a message.
7. You want to complete a Diffie-Hellman key exchange with Bob. Suppose your communications channel is controlled by an adversary, Eve, who can block messages between you and Bob and resend them, pretending to be you. Is it possible for Eve to disrupt the key exchange and make it appear as though you and Bob have a common secret, when really it is Eve who shares a different common secret with each of you?

Chapter 24

RSA

We alluded to the idea of a public-key cryptographic system when discussing Diffie-Hellman key exchange, but did not explain what such a system is or how it works. Recall that the cryptographic systems we have discussed so far have been symmetric; that is, the recipient and sender have a shared key that they use for encryption and decryption in some way. The problem of transmitting such a key is troublesome, as we have seen. Diffie-Hellman key exchange helps this problem, but has problems of its own. In particular, it is vulnerable to a man-in-the-middle attack and requires both parties to compute a shared key at the same time.

In 1977, cryptographers Ronald Rivest, Adi Shamir, and Leonard Adleman developed a cryptographic system, RSA, for which no shared key is required. Each user of the RSA system generates two related keys, one public and one private. These keys have the following properties:

- Any message encrypted with a public key can only be decrypted by the corresponding private key.
- Any message encrypted with a private key can only be decrypted by the corresponding public key.
- Given any public key, there is no efficient way to determine the corresponding private key.
- Given any private key, there is no efficient way to determine the corresponding public key.

The basic idea behind public-key cryptography was described by Diffie, Hellman, and Merkle, though they were unable to provide an example of such a system.

Before we describe the specifics of the RSA algorithms, we will explain how such a system could be used for secure communication. Suppose that Alice wants to send a secret message to Bob using RSA. We assume that both Alice and Bob have generated their own public and private RSA keys, and that the public keys are known to everyone. Alice writes her message and encrypts it using Bob's public key. She then sends the encrypted message to Bob. Since Bob is the only person with the correct private key, he can decrypt the message. If Eve is eavesdropping on the communication channel, she cannot decrypt the message since she does not have Bob's private key and cannot efficiently determine it from his public key.

To explain how the RSA algorithms work, we need some mathematical tools first.

24.1 Totient function

It will be useful to know, for a given number n, how many positive integers are coprime to n. In a sense, by counting these, we are somehow measuring "how prime" the number n is.

Definition 24.1. The *totient* or *Euler function* is the function ϕ such that $\phi(n)$ gives the number of positive integers less than n that are coprime to n.

Since one is coprime to every integer, $\phi(n)$ is always a positive number. When n is small, we can compute $\phi(n)$ directly.

Example 24.2. In the case when $n = 10$, the positive integers coprime to n are the integers in the set $\{1, 3, 7, 9\}$. Hence $\phi(10) = 4$.

However, there is no efficient way to calculate $\phi(n)$ when n is very large. When n takes on certain special forms, though, $\phi(n)$ may be very easy to compute.

Fact 24.3. *If p is a prime number, then $\phi(p) = p - 1$.*

Proof. The only positive factors of p are 1 and p. Hence, every positive integer less than p is coprime to p. There are $p - 1$ such integers. □

In some cases, understanding the factorization of an integer can help determine the value of its totient.

Fact 24.4. *If m and n are coprime, then $\phi(mn) = \phi(m)\phi(n)$.*

This fact does not hold in general if the two integers are not coprime. We will see additional facts like this in the exercises.

24.2 Euler's theorem

A result of Euler is needed for the RSA algorithms. It states the following:

Theorem 24.5 (Euler's theorem)**.** *If n is any integer and $\gcd(a, n) = 1$, then $a^{\phi(n)} \equiv 1 \bmod n$.*

Example 24.6. Consider $n = 26$ and $a = 11$; clearly $\gcd(a, n) = \gcd(11, 26) = 1$, so Euler's theorem applies. Observe that $\phi(n) = \phi(26) = 12$, and that $a^{\phi(n)} = 11^{12} \equiv 1 \bmod 26$ as expected.

Note that we can deduce Fermat's little theorem from Euler's theorem. If p is prime, then $\phi(p) = p - 1$, so $a^{p-1} \equiv 1 \bmod p$ and hence $a^p \equiv a \bmod p$.

24.3 Key generation

Recall that in order to send and receive encrypted messages using RSA, we need a public and private key. In order to do so, we begin by choosing very large primes p and q, and computing their product $n = pq$. In practice, n has hundreds or thousands of bits in its binary representation. Note that we can immediately compute $\phi(n) = (p - 1)(q - 1)$ as in the exercises.

We then find an integer e such that $1 < e < \phi(n)$ and e is coprime to $\phi(n)$. In practice, e is usually chosen to be the prime integer 65537. The numbers n and e comprise our public key. We publish this key freely, but are careful never to release the factors p or q.

We then compute $d \equiv e^{-1} \bmod \phi(n)$. This is easy to do using the extended Euclidean algorithm. The numbers n and d comprise our private key. We do not publish d.

Notice two key observations about key generation.

- Given the value of n, an attacker Eve cannot efficiently compute $\phi(n)$ without knowing its factors. However, factoring large numbers is very difficult.
- Given the public key e, Eve cannot efficiently compute $e^{-1} \bmod \phi(n)$ since she does not know $\phi(n)$. Hence it is infeasible to determine a private key from the corresponding public key (or vice versa).

Example 24.7. Suppose we want to generate an RSA keypair. The numbers we use will be very small for the purposes of this example to make computations easier to do by hand. We choose primes $p = 61$ and $q = 53$; then $n = pq = 3233$ and $\phi(n) = 60 \times 52 = 3120$. To generate our public key, we choose $e = 17$ and note that $\gcd(e, \phi(n)) = 1$. To generate our private key, we compute $d \equiv e^{-1} \bmod \phi(n) \equiv 2753 \bmod 3120$.

This means our public key is the pair $(n, e) = (3233, 17)$ and our private key is the pair $(n, d) = (3233, 2753)$.

24.4 Encryption

Suppose that Alice wants to send an encrypted RSA message to Bob. She knows Bob's public key e and the chosen modulus n. She converts her message into an integer M that is less than n. There are many schemes for doing so. She then computes $C \equiv M^e \bmod n$ and transmits this ciphertext message C to Bob. Note that she can efficiently compute C using, for example, the square-and-multiply method, even when e is very large.

24.5 Decryption

Once Bob receives the ciphertext message C, he computes $C^d \bmod n$. We need to show that this recovers the plaintext message M. First note that since e and d are inverses $\bmod \phi(n)$, we have $ed \equiv 1 \bmod \phi(n)$. By definition, this means $\phi(n)$ divides $ed-1$; that is, there is some integer h such that $ed - 1 = h\phi(n)$.

Bob computes the following:

$$\begin{aligned} C^d \bmod n &\equiv (M^e)^d \bmod n \\ &\equiv M^{ed} \bmod n \\ &\equiv M^{h\phi(n)+1} \bmod n \\ &\equiv (M^{\phi(n)})^h M \bmod n \\ &\equiv 1^h M \bmod n \\ &\equiv M \bmod n \end{aligned}$$

Note that we concluded that $M^{\phi(n)} \equiv 1 \bmod n$ from Euler's theorem. There is a small technicality that we need $\gcd(M, n) = 1$, but in practice this is not an issue.

24.6 Exercises

1. Let p and q be distinct primes. Show that $\phi(pq) = (p-1)(q-1)$.
2. Show that $\phi(p^k) = p^k - p^{k-1}$ if p is prime and k is a positive integer. Give a version of Fermat's Little Theorem for which the modulus is of the form p^k.
3. Find $\phi(12)$.
4. Find $\phi(36)$. Show that $\phi(36) \neq \phi(6)\phi(6)$, but that $\phi(36) = \phi(4)\phi(9)$.
5. Let $n = 91$, which is the product of two distinct primes. Choose a message M that is coprime to n. Perform the RSA encryption and decryption algorithms to ensure you recover M.
6. Suppose that I want to send you a message that has been encrypted with my private key. The message is not secret, but I want to make sure you can verify that I wrote it. I send you my public key (where $n = 26025927$ and $e = 13$) and an encrypted message of the form $M^e = 16961512$. Find the plaintext message M (a two-digit almost wonderful number).
7. For this RSA exercise, your public key consists of the numbers $e = 455$ and $n = 851 = 23 \times 37$.
   (a) Compute $\phi(n)$.
   (b) Find your private key d by computing $e^{-1} \bmod \phi(n)$.
   (c) You receive the ciphertext message $C \equiv_n 20$. Decrypt it by computing $C^d \bmod n$.
8. Find a number x such that $2^x \equiv 1 \bmod 21$. Use exponentiation to verify this.
9. For each step of the RSA process (key generation, key distribution, encryption, message distribution, decryption), discuss what an adversary would need to know or determine in order to break that part of the algorithm.

10. Suppose you decide to use a 512-bit value of N for RSA encryption. Suppose an adversary tries to break the algorithm by trying to factor N; that is, they will try to divide N by every positive integer less than $\sqrt{N}$. If the adversary can test ten billion integers per second, how long will it take to guarantee that N is factored? Why is it sufficient to only check positive integers up to $\sqrt{N}$?

11. Suppose that Eve can actively interfere with the communication channel used by Alice and Bob while they are distributing their public keys and sending messages. Explain how Eve can read messages sent from Alice to Bob while still passing the messages on to Bob.

Chapter 25

ElGamal

Although RSA was the first public-key cryptosystem, it is not the only one. Another scheme, developed by Taher ElGamal in 1985, is based on the principles of Diffie-Hellman key exchanged discussed earlier.

In order for Bob to send Alice a message using ElGamal, Alice must first establish her public and private keys. Alice and Bob are assumed to have already agreed upon a large prime p and an integer $g \bmod p$ that has large prime order. Alice then chooses a random $a \bmod p$ as her private key, and publishes $A \equiv g^a \bmod p$ as her public key.

Bob prepares his message as an integer $m \bmod p$. However, he also chooses another random integer $k \bmod p$, called an *ephemeral key*, that is used only for this particular encryption. Bob then computes two values that comprise the ciphertext: $c_1 \equiv g^k \bmod p$ and $c_2 \equiv mA^k \bmod p$. Note that Bob can easily compute both of these values, since g and p are previously agreed upon and Alice has published her public key A. Bob then transmits c_1 and c_2 to Alice (and Eve, if she is listening in on the communication channel).

To decrypt the message, Alice first uses her private key to compute the value $x \equiv c_1^a \bmod p$ and, using the extended Euclidean algorithm, its inverse $x^{-1} \bmod p$. She then recovers the plaintext by computing the following:

$$\begin{aligned} x^{-1}x_2 &\equiv (c_1^a)^{-1}x_2 \bmod p \\ &\equiv (g^{ak})^{-1}c_2 \bmod p \\ &\equiv (g^{ak})^{-1}mA^k \bmod p \\ &\equiv (g^{ak})^{-1}mg^{ak} \bmod p \\ &\equiv m \bmod p \end{aligned}$$

25.1 Comparison to RSA

We now note key differences between ElGamal and RSA.

Unlike RSA, where security is based on the assumption that it is difficult to factor large numbers into prime factors, the security of ElGamal is based on the assumption that the discrete logarithm problem is difficult to solve.

Recall that for a given message that Bob transmits to Alice, there is exactly one resulting ciphertext. This is because Bob only uses Alice's public key to determine how to encrypt the message. By contrast, in the ElGamal scheme, the use of an ephemeral key means that a given message from Bob to Alice can have many possible ciphertexts. We say that RSA is a *deterministic* cryptosystem, while ElGamal is a *probabilistic* cryptosystem. The advantage of a probabilistic scheme is that an attacker cannot rely on identical messages having identical ciphertexts to gain information about communications.

Further, unlike RSA, each ElGamal ciphertext has two pieces of information required for transmission. This means that in general, RSA is a more compact scheme.

Chapter 26

Hash functions and digital signatures

For many applications in cryptography and beyond, it is useful to work not with a piece of data, but with some sort of representation of the data. The most common way to do this is through the use of *hash functions*. Essentially, a hash function takes in any amount of data and spits back some kind of representation of that data, generally in a more compact form. However, such functions generally need to possess a few important properties.

Definition 26.1. A *hash function* is a function f that, given any amount of data, outputs a representation of that data of fixed length such that:

- given some input x, it is efficient to compute $f(x)$
- given some output $f(x)$, it is not efficient to determine the original input x
- given different inputs x and y, it is not likely that $f(x) = f(y)$

That is, we require that f be *efficient*, *one-way*, and *collision-resistant*.

It is important to understand the difference between hashing and encryption. When we encrypt data, it is possible to recover it given the correct key. When hashing data, it is not possible to recover it.

Before we see examples of hash functions, let's examine their purpose. Why would such a function be useful in practice?

- Password hashing. When you create an account with an online service, the service hashes the password you give it and only stores the hash, not the password. When you log in, the service hashes the password you provide and compares this hash to the one it has on file. If they match, collision-resistance ensures you likely entered the correct password. The fact that the hash function is one-way ensures an attacker cannot determine your password if the service is compromised.
- File integrity. When large files are transferred, the sender typically hashes the file and sends a copy of the hash to the recipient. When the recipient receives the file, she hashes it and compares it to the hash she received. If they match, it is likely there were no errors during the transfer. If they do not match, she can request the file again from the sender.
- Digital currency. Some digital currencies, like Bitcoin, rely on hashes to ensure the integrity of financial transactions. This type of use is more complicated, so we will not discuss it in detail here.
- Digital signatures. We will discuss this use in more detail shortly.
- Message authentication codes. We will discuss this in detail later.

It is possible to construct our own simple hash function, but difficult to do in a way that provides the three desired properties. One example is to XOR all bits in a binary message, and use the result as its hash. Let's call this the *parity hash*. Does it possess the desired properties? Note that it certainly takes in messages of any length and outputs a fixed representation (of one bit).

- Efficient? The XOR operation is very fast to implement in hardware or software, and even to do by hand. The parity hash passes!

- One-way? Given only the output of the parity hash (either a one bit or a zero bit), there are infinitely many messages that hash to that value. It is extremely unlikely to pick the correct one. The parity hash passes!

- Collision-resistant? Notice that any two messages with either an even number of one bits or an odd number of one bits will hash to the same value. The parity hash fails horribly!

For many years, the MD5 hash was used extensively for the applications listed above. However, a series of weaknesses was found in the algorithm over time, culminating in methods for explicitly generating collisions. The algorithm is now considered broken. A family of hashes called the Secure Hash Algorithm (SHA) functions was developed as a replacement. A full collision in the hash known as SHA-1 has been found. Even so, its successors, the SHA-2 family, are considered safe and are now widely used. An open contest for a next generation SHA-3 family of algorithms resulted in even better functions.

26.1 Merkle-Damgård transform

Many hash functions, like SHA-1 and MD5, operate internally on small blocks of data at a time. However, we know that a hash function must accept input of any length. Hence, we need a way to take functions operating on fixed amounts of data and transform them into functions operating on any length of data. The method we describe here, called the *Merkle-Damgård transform*, is the approach used by both SHA-1 and MD5.

For this construction, suppose we have a function h that takes n bits of data and hashes them to $n/2$ bits (where n obviously must be even). We take our message m of L bits and break it up into blocks $m_1, \ldots, m_b$ of length $n/2$. If needed, we pad the last block with zeros so it is the correct length. We iteratively apply the function h according to Figure 26.1, using the output from h along with the next block of m at each step. At the last step, we input the length L of the message for technical reasons. Notice that because h takes input of n bits, we are always putting in the correct number of bits at each step. We typically call this new hash function H, so the output is denoted $H(m)$.

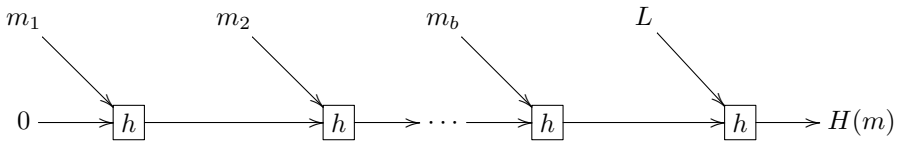


Figure 26.1: Diagram of the Merkle-Damgård transform

26.2 Sample hash function

To demonstrate hash techniques, we will define our own, the Cryptographic Hash Function (CHF).

Let the message be $M = a_0a_1 \cdots a_{29}$. Then $\text{CHF}(M) = b_0b_1 \cdots b_9$ for the following output bits:

$$\begin{aligned}
b_0 &= a_0 \oplus a_6 \oplus a_{12} \oplus a_{18} \oplus a_{24} \oplus a_3a_4 \\
b_1 &= b_0a_1 \oplus a_6 \oplus a_{11} \oplus a_{16} \oplus a_{21} \oplus a_{26} \\
b_2 &= b_1a_2 \oplus a_6 \oplus a_{10} \oplus a_{14} \oplus a_{18} \oplus a_{22} \oplus a_{26} \\
b_3 &= b_2a_3 \oplus a_6 \oplus a_9 \oplus a_{12} \oplus a_{15} \oplus a_{18} \oplus a_{21} \oplus a_{24} \oplus a_{27} \\
b_4 &= b_3a_4 \oplus a_{10} \oplus a_{16} \oplus a_{22} \oplus a_{28} \\
b_5 &= b_4a_5 \oplus a_{10} \oplus a_{15} \oplus a_{20} \oplus a_{25} \\
b_6 &= a_0 \oplus a_1 \oplus a_2 \oplus a_3 \oplus a_4 \oplus a_5 \oplus a_6 \oplus a_7 \oplus a_8 \oplus a_{13} \oplus a_{14} \oplus a_{28} \\
b_7 &= b_6a_7 \oplus a_{11} \oplus a_{15} \oplus a_{19} \oplus a_{23} \oplus a_{27} \\
b_8 &= b_7a_8 \oplus a_{11} \oplus a_{14} \oplus a_{17} \oplus a_{20} \oplus a_{23} \oplus a_{26} \oplus a_{29} \\
b_9 &= b_8a_9 \oplus a_{14} \oplus a_{19} \oplus a_{24} \oplus a_{29}
\end{aligned}$$

For longer messages, break up into padded 30-bit blocks and XOR the resulting outputs together.

26.3 Digital signatures

We can combine public-key cryptography with hash functions in a powerful way. We have already seen that Alice can use public-key cryptography to send a message that only Bob can read. She did this by encrypting her message with Bob's public key; Bob then decrypts with his private key. This solves the problem of *secrecy*. There is a problem with this arrangement, however. How does Bob know that Alice wrote the message? Eve can easily write a message pretending to be Alice and encrypt it with Bob's public key. This is the problem of *authentication*; we need a way to verify that a message is from Alice and not Eve.

Suppose that Alice tries something different. Suppose she encrypts her message instead with her private key. This means only Alice's public key can decrypt the message. This is obviously not suitable for encrypting a message to Bob since everyone has access to Alice's public key. However, this arrangement solves the authentication problem! If Bob wants to make sure the message came from Alice, all he needs to do is use her public key to decrypt the message. There is no way for Eve to forge a message from Alice, since she would need access to Alice's private key to do so, which she does not.

Notice that this arrangement makes Alice's message essentially public, since Eve can decrypt it too. What if Alice wants to send Bob a secret message that he can verify came from her? We can combine our two public-key techniques to accomplish both secrecy and authentication. Alice first encrypts her message with her private key, and then with Bob's public key. Bob uses his private key to decrypt the first "layer" of the message, and then Alice's public key to ensure the resulting message came from her. Eve cannot read the message since she does not have Bob's private key.

None of this yet involves hashes. The benefit of hashes becomes clearer when we notice that public-key algorithms are typically very slow and computationally intensive. Instead of using her private key to encrypt her entire message, she instead hashes the message encrypts only the hash with her private key. Anyone who wants to authenticate the message decrypts the hash they receive, hashes the message, and compares the two. If they are the same, then Alice sent the message. This technique is called a *digital signature*.

26.4 Exercises

1. Compute the CHF hash value of the word ASTLEY, using the standard 29-letter binary representation.

2. See if you can find a message that hashes to each of the following values. What does this tell us about the CHF hash function?

   (a) 0x000

   (b) 0x1F4

3. Choose any 30-bit message and compute its CHF hash value. Change one bit in the message and determine how this affects the resulting CHF hash value.

4. Suppose that Alice signs a digital document and Bob receives the document and signature.

   (a) How does Alice sign the document?

   (b) How does Bob verify the signature?

   (c) Why does Bob know that the document was not altered after Alice signed it?

   (d) Why does Bob know that Eve did not send a phony signed document in place of Alice's?

5. For this exercise, you will generate your own hash function that takes ten bits as input (that is, two characters) and outputs five bits (that is, one character).

   (a) Suppose the input bits are labeled $a_1a_2a_3\cdots a_{10}$. Determine formulas that give each of the five output bits $b_1b_2b_3b_4b_5$ in terms of the input bits and previously computed output bits, using either XOR or multiplication operations. For example, you might let $b_3 = b_2a_1 \oplus b_1a_2 \oplus a_4 \oplus a_8$. You should have one formula for each of the five output bits.

   (b) Compute the hash value of your initials.

   (c) Compute the hash values of `AF` and `AH`, which differ by a single bit. In how many bits do the resulting hash values differ?

   (d) Find two characters of input that hash to the output `00000` = `A`.

Chapter 27

Block ciphers

Recall that a linear feedback shift register is an example of a stream cipher, since it generates key bits that encrypt one plaintext bit at a time. In theory, an LFSR can encrypt any amount of data, though we have already seen that there must be repetition in the key stream. In contrast, *block ciphers* operate on chunks of data at a time. These chunks are called *blocks*.

Definition 27.1. A *block cipher* is a cipher that encrypts a fixed number of bits at a time. It takes as input n bits (called a *block*) and a key of k bits, and outputs n bits of ciphertext. In this case, k is called the *key length* and n the *block size*.

Example 27.2. We have already seen an example of a block cipher. The Playfair cipher is a block cipher with a key length of five characters (used to construct the key grid) and a block size of two characters.

Often, a block cipher will use a simple operation, called a *round function*, on a block of plaintext data and iterate the procedure several times. Each iteration of the round function is called a *round*.

27.1 Feistel ciphers

A particular and common type of block cipher is called a *Feistel cipher* or *Feistel network*, named after cryptographer Horst Feistel of IBM. Feistel cipher apply a particular sequence of operations to blocks of data in multiple rounds. The initial step of Feistel encryption is to break plaintext data into two equal halves. A sequence of swapping and mixing, called a *round*, is continued for as long as specified by the particular cipher.

Suppose we want to create a Feistel cipher with a block size of b bits, where b is even, and n rounds. The cipher requires a function F, called the *round function*, that takes a key and $b/2$ bits of data as input and spits out $b/2$ bits as output. The cipher requires n so-called *subkeys* $K_0, \ldots, K_{n-1}$, one for each round.

Encryption follows the following algorithm:

1. Split the input block into two halves (L_0, R_0).
2. For each round i, let $L_{i+1} = R_i$ and $R_{i+1} = L_i \oplus F(R_i, K_i)$.
3. The ciphertext block is (R_n, L_n); that is, we do not swap halves after the last round.

Decryption follows the following algorithm:

1. Split the input block into two halves (R_n, L_n). The order is reversed here to simplify the notation.
2. For each round i, let $R_i = L_{i+1}$ and $L_i = R_{i+1} \oplus F(L_{i+1}, K_i)$.
3. The plaintext block is (L_0, R_0); that is, we do not swap halves after the last round.

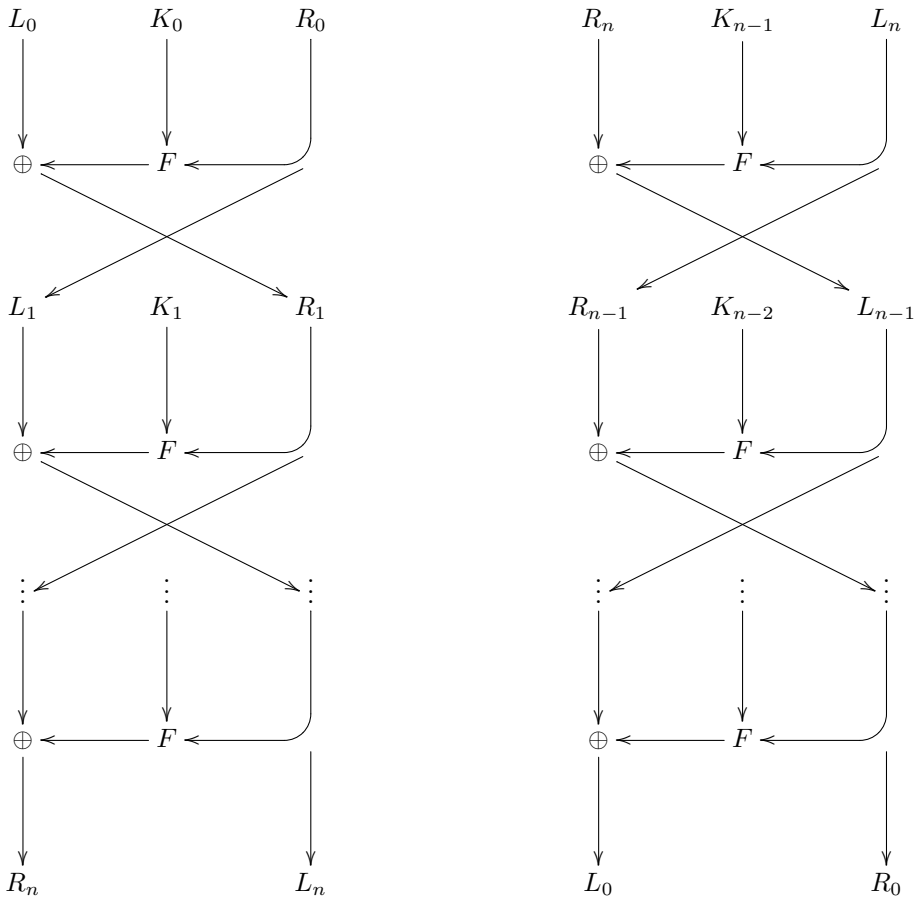


Figure 27.1: Feistel encryption (left) and decryption (right) algorithms

The algorithms for encryption and decryption are easily seen using diagrams. Figure 27.1 demonstrates the encryption and decryption algorithms for Feistel ciphers.

At first glance, the decryption algorithm seems questionable. After all, most of the ciphers we have dealt with required the encryption methods to be reversible; otherwise, decryption was ambiguous. But we made no such requirement on our round function F. Why should we expect a non-reversible function to result in a reversible cryptographic system?

Proposition 27.3. *Regardless of the choice of round function, every Feistel cipher is reversible.*

Proof. We will show that each round of encryption has a corresponding round of decryption, making the entire Feistel cipher reversible. Suppose we encrypt the plaintext block (A, B) in a given round; the ciphertext block is $(B, A \oplus F(B, K))$, where K is the round subkey. If we apply one round of decryption to this ciphertext, we obtain $(A \oplus F(B, K) \oplus F(B, K), B) = (A, B)$ and recover the plaintext block. □

Example 27.4. Let's construct an example of a Feistel cipher. Our cipher will have a block length of ten bits, three rounds, and subkeys of five bits each. Our round function must operate on half the block length, so we will choose the function

$$F(x_1x_2x_3x_4x_5, k_1k_2k_3k_4k_5) = k_1 + (x_1 + x_2), k_2 + (x_2x_3), k_3 + (x_3 + x_4), k_4 + (x_4x_5), k_5 + (x_5 + x_1)$$

where all sums and products are taken $\bmod 2$, $x_1x_2x_3x_4x_5$ is the input data, and $k_1k_2k_3k_4k_5$ is the subkey. This choice of round function is arbitrary; it does not need to have any particular form.

Let's encrypt the block (that is, digraph) "no" using this Feistel cipher. If we encode our message using the standard representation, it becomes `01101 01110`. We will choose our three subkeys by using a larger key of three characters and splitting it into three equal parts; suppose we choose "foo" as the key, which encodes to `00101 01110 01110`. The three encryption rounds proceed as follows:

Round 0:	`01101 01110`
Round 1:	`01110 10000`
Round 2:	`10000 10001`
Round 3:	`10001 01110`
Ciphertext:	`01110 10001`

This gives the ciphertext digraph `OR`. Note that in the decryption, we swap the output of the last round according to the algorithm.

To verify our encryption, we apply the decryption algorithm to the resulting ciphertext block, using the same subkeys in reverse order. The three decryption rounds proceed as follows:

Round 3:	`01110 10001`
Round 2:	`10001 10000`
Round 1:	`10000 01110`
Round 0:	`01110 01101`
Plaintext:	`01101 01110`

This recovers the plaintext block.

27.2 Data Encryption Standard

The most well-known block cipher is the Data Encryption Standard (DES). The DES cipher, based on an earlier block cipher developed at IBM, was established as a United States government standard for data encryption in 1976. The DES cipher has a block size of 64 bits and a key length of 56 bits. It is an example of a Feistel cipher, though its round function is complicated.

When creating DES, IBM consulted with the U.S. National Security Agency (NSA) for guidance. The NSA reduced the DES block size and its key length, and also altered substitution elements of the design. Despite worries that these alterations weakened the algorithm, later analysis revealed that the NSA had in fact strengthened the DES algorithm against possible attacks.

Since the key length of DES is so small, advances in computing eventually resulted in it being possible to break DES ciphertext by exhausting all possible decryption keys, often in under a day; the first public break occurred in 1997 after a $10,000 contest was announced by RSA Security. The cipher was reaffirmed as a federal standard in 1983, 1988, 1993, and 1999; however, in 1999, a variant called triple DES was suggested instead. DES was officially withdrawn as a standard in 2005, though triple DES remains approved for unclassified government use through 2030.

27.2.1 BabyDES

To help understand the round function and operation of DES, we will study a simplified version (created for this purpose) that we will call *BabyDES*. It contains most of the basic operations of DES, but with large operations reduced to a size more manageable to do by hand. There are a few small and less-important elements of the DES algorithm, like key preparation, that have been removed from BabyDES to save time. We will introduce the algorithm, perform an example throughout the process, and then compare BabyDES to DES more directly.

BabyDES, like DES, is a Feistel network. This means it takes a message block, splits it, applies a round function that involves a subkey, performs an XOR operation, and swaps block halves. This process is repeated for each round. In this case, BabyDES is defined to have three rounds, require a key of ten bits, and encrypt or decrypt a ten-bit message block. Using our standard binary representation for letters, this means that each message block and key are comprised of two characters.

As an example, we will encrypt the plaintext message block "OK" (binary `01110 01010`) using the key "FL" (binary `00101 01011`) with BabyDES.

27.2.2 Subkeys

Each of the three BabyDES rounds requires a subkey. We form these subkeys by performing successive left shifts of the key and taking the first eight bits of the result:

Subkey	Binary
K_0	0010 1010
K_1	0101 0101
K_2	1010 1011

As with any Feistel network, each of the subkeys is used in order for encryption and in reverse order for decryption. Always pay attention to which order is needed!

27.2.3 Round operation

We will describe the round operation, including the round function, using the example. First, split the message block into halves L_0 and R_0:

$$\begin{aligned} L_0 &= \texttt{01110} \\ R_0 &= \texttt{01010} \end{aligned}$$

The round function begins now. The right half-block (R_0 in the first round) is expanded to eight bits using the order $(1, 2, 4, 3, 4, 3, 5, 2)$:

$$\texttt{01010} \to \texttt{0110 1001}$$

The subkey and expanded right half-block are XORed together:

$$K_0 \oplus R_0 = \texttt{0010 1010} \oplus \texttt{0110 1001} = \texttt{0100 0011}$$

The next step is to apply a substitution using a construction called an *S-box* (for substitution box). Basically, the result of the XOR operation is used to look up a new binary value in an S-box grid. There are two S-boxes in BabyDES, where

$$S_1 = \begin{pmatrix} 101 & 010 & 001 & 110 \\ 011 & 100 & 111 & 000 \\ 001 & 100 & 110 & 010 \\ 000 & 111 & 101 & 011 \end{pmatrix}$$

and

$$S_2 = \begin{pmatrix} 10 & 00 & 11 & 01 \\ 11 & 00 & 01 & 10 \\ 01 & 11 & 00 & 10 \\ 00 & 10 & 01 & 11 \end{pmatrix}.$$

Recall that the XOR output was 0100 0011. The first half of the output is used to look up three bits from S_1, using the first two bits for the row and the second two bits for the column. So the S_1 output, from the second row and first column, is 011. Similarly, we get two bits from S_2 using the second half of the output for the row and column. So the S_2 output, from the first row and fourth column, is 01.

The outputs from S_1 and S_2 are concatenated together in order to give, at long last, the output of the round function. In this case, the round function output is 01101.

As in any Feistel network, we take the left half-block and XOR it with the round function output:

$$\texttt{01110} \oplus \texttt{01101} = \texttt{00011}$$

To enter the second round, we perform a swap:

$$\begin{aligned} L_1 = R_0 &= \texttt{01010} \\ R_1 &= \texttt{00011} \end{aligned}$$

We then proceed with the second and third rounds, remembering that we do not swap after the last round (to ensure proper decryption later). For completeness, here are the results at the end of each round, listed after the swap:

Round 0:	`01110 01010`
Round 1:	`01010 00011`
Round 2:	`00011 01001`
Round 3:	`01001 10110`
Ciphertext:	`10110 01001`

This gives the ciphertext digraph `TH`.

27.2.4 Extending BabyDES to DES

In the actual DES algorithm, each block is 64 bits, meaning the round function operates on a 32-bit half-block. The key length is 64 bits, of which only 56 bits are actually used. There are 16 rounds, each of which uses a subkey of 48 bits. The process used to form the subkeys has several steps that involve permutations and shifts; BabyDES uses only a simplified version of this process.

Prior to any rounds occurring, there is an initial permutation of the message block; there is a corresponding reverse permutation after the final round. These permutations have no cryptographic purpose, but were intended to make it easier to load and unload data in hardware at the time.

The round function of DES begins with an expansion of the 32-bit half-block to 48 bits. This expansion is XORed with the subkey, as in BabyDES. Then, the XOR output is used for lookups in eight different S-boxes, each of which uses six bits as lookup and outputs six bits. The output of all S-boxes is concatenated together, and a permutation (called a *P-box*) is applied to help distribute the output more broadly in the next round.

27.3 Advanced Encryption Standard

To replace DES, the U.S. National Institute of Standards and Technology held an open contest for adoption of a new government standard, the Advanced Encryption Standard (AES). Fifteen ciphers were subject to a five-year analysis process, after which the Rijndael family of ciphers was chosen in 2001. Unlike DES, AES is publicly approved to encrypt top-secret government data in the United States.

The block size for AES is a fixed 128 bits; however, keys may be of 128, 196, or 256 bits in length. It is not a Feistel cipher. No known complete attacks exist for AES with any key length.

27.4 Other block ciphers

There are other modern block ciphers used less frequently than, for example, AES. Blowfish is a Feistel cipher designed by Bruce Schneier in 1993; it has a block size of 64 bits and a key length variable from 32 to 448 bits. Twofish is a Feistel cipher designed as a successor to Blowfish by Schneier in 1998 and submitted as an AES candidate; it has a block size of 128 bits and a key length of either 128, 192, or 256 bits. Neither cipher has been fully broken; in fact, Twofish remains an option in the popular OpenPGP standard.

27.5 Block cipher modes

How would we encrypt a message longer than one block using a block cipher? After all, block ciphers can only act on a single block at a time. Methods for using a block cipher to encrypt longer messages are called *block cipher modes*.

27.5.1 Electronic code book (ECB) mode

A simple block cipher mode, the *electronic code book* (ECB) cipher mode, takes a message of l blocks and encrypts each block using the same block cipher F_k using secret key k. Figure 27.2 diagrams the ECB encryption of a message $m_1 m_2 \cdots m_l$ and the resulting ciphertext $c_1 c_2 \cdots c_l$.

Though it is simple and easy to implement, ECB has a fundamental flaw. To see it, suppose we use a four-character block cipher that uses ECB. If we encrypt the two plaintext messages “starbuck” and “startnow”

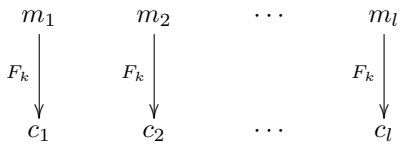


Figure 27.2: Electronic code book block cipher mode encryption

using this cipher, each ciphertext message will have two blocks since each plaintext messages has two blocks. However, note that each message has "star" as its first block. Since ECB uses the same key for each block, this means the two ciphertext messages will start with the same four characters. A cryptanalyst could use information like this to determine the nature of other ciphertext messages.

This flaw in ECB is not trivial. Adobe Systems, a software corporation in the United States, used a block cipher (likely triple DES) to encrypt over 150 million of its users' passwords using the same unknown key. This database also included an unencrypted password hint provided by the user for each of the entries. The database was stolen in October 2013 and released to the public by the attackers. While the key is still not known, researchers have been able to obtain significant information about the passwords in the database by examining identical ciphertext blocks and using the corresponding password hints to determine passwords. An investigation of this attack appears in the exercises.

27.5.2 Cipher block chaining (CBC) mode

In the *cipher block chaining* (CBC) block cipher mode, encryption requires two components, the key and an additional block of random data called an *initialization vector*. The initialization vector is used in the first round of encryption and ensures that identical messages with unique initialization vectors do not encrypt identically. The first encrypted block is then used as the initialization vector for the second encryption round, and so on. The initialization vector is transmitted as the "zeroth block" of ciphertext. Figure 27.3 diagrams the CBC encryption of a message $m_1 m_2 \cdots m_l$ using initialization vector IV and the resulting ciphertext $IV c_1 c_2 \cdots c_l$.

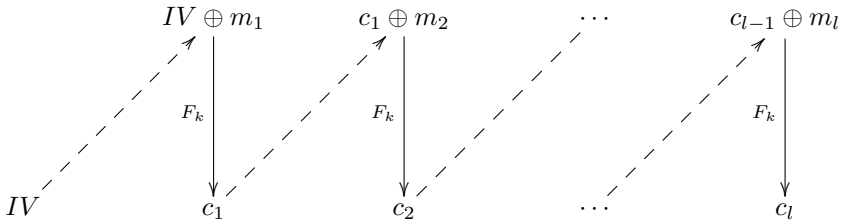


Figure 27.3: Cipher block chaining block cipher mode encryption

27.5.3 Output feedback (OFB) mode

The *output feedback* (OFB) block cipher mode, a block cipher F_k is used with an initialization vector to generate a stream of bits that is XORed with a plaintext message to generate ciphertext. This essentially transforms a block cipher into a stream cipher. Figure 27.4 diagrams the generation of an OFB bit stream $s_1 s_2 \cdots s_l$, encryption of a message $m_1 m_2 \cdots m_l$ with an initialization vector IV, and the resulting ciphertext $IV c_1 c_2 \cdots c_l$. As in CBC encryption, the initialization vector forms the "zeroth block" of ciphertext.

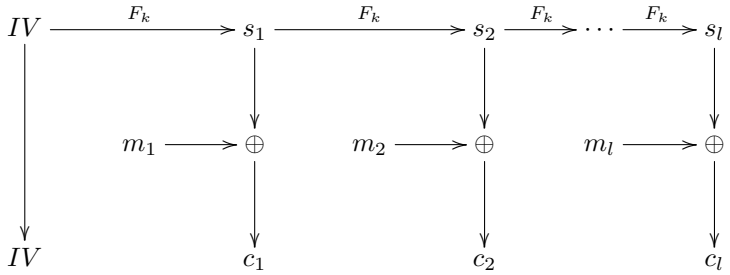


Figure 27.4: Output feedback block cipher mode encryption

27.5.4 Counter (CTR) mode

Like output feedback, the counter (CTR) block cipher mode generates a stream of bits that is XORed with a plaintext message to generate ciphertext, effectively turning a block cipher F_k into a stream cipher. This cipher mode first XORs an initialization vector with an increasing counter as the input to F_k for successive blocks. Figure 27.5 diagrams the generation of a CTR bit stream $s_1 s_2 \cdots s_l$, encryption of a message $m_1 m_2 \cdots m_l$ with an initialization vector IV, and the resulting ciphertext $IV c_1 c_2 \cdots c_l$. As in OFB encryption, the initialization vector forms the "zeroth block" of ciphertext.

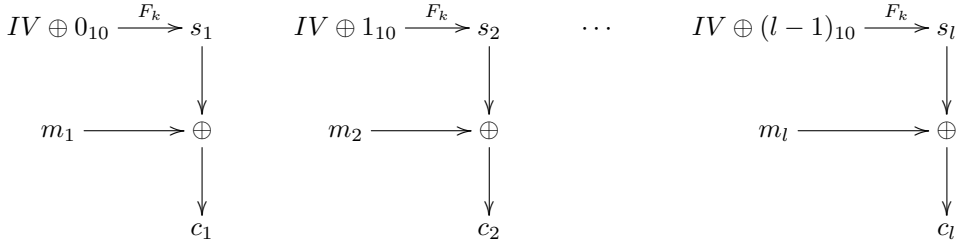


Figure 27.5: Counter block cipher mode encryption

27.6 Exercises

1. Draw a diagram for ECB decryption. Suppose F_k^{-1} denotes the corresponding decryption function.
2. Draw a diagram for CBC decryption. Suppose F_k^{-1} denotes the corresponding decryption function.
3. Draw a diagram for OFB decryption. Why is it not necessary that the block cipher F_k have a corresponding decryption function?
4. Draw a diagram for CTR decryption. Why is it not necessary that the block cipher F_k have a corresponding decryption function?
5. For each of the four block cipher modes, determine which of encryption, decryption, or both are parallelizable.
6. Suppose we have a Feistel cipher that has a block size of ten bits (that is, two characters) and a key size of twenty bits (that is, four characters). This particular cipher runs four rounds with the following

round function:

$$F(k_1k_2k_3k_4k_5, b_1b_2b_3b_4b_5) = \begin{cases} k_1 \oplus b_5 \\ k_2 \oplus b_4 \\ k_3 \oplus b_3 \\ k_4 \oplus b_2 \\ k_5 \oplus b_1 \end{cases}$$

The ciphertext message `NSVO` was obtained using ECB mode encryption with the key ABCD. Decrypt the message.

7. Suppose a Feistel cipher has a round function that outputs all zeros. Use Figure 27.1 to explain what effect this has on ciphertext. Be sure to consider cases when the network has an even or odd number of rounds.

8. Suppose a Feistel cipher has a round function that leaves the input unchanged. Use Figure 27.1 to explain what effect this has on ciphertext. Be sure to consider all possible cases.

9. Using BabyDES with the key "KE", decrypt the ciphertext block `JE`.

10. This exercise simulates the Adobe Systems password breach discussed earlier. In this example, data similar to that obtained from Adobe's password database is provided. Each entry contains an encrypted password broken into blocks and the corresponding password hint supplied by the user who chose the password. Assume the encryption was performed using a block cipher in ECB mode with a four-character block size, and that plaintext passwords all have a length of four or eight. Determine each password.

Encrypted password	Password hint
4991 8080	too easy
1208 6344	these animals eat these animals
0086 3132	my favorite six films
0086 9437	coffee, or ahab's mate
9437	the ___ stops here
1113 4321	poppycock
1008 4321	change from gas to liquid
0410 6344	bowl animal, or cracker
5562 7248	month
1118 7248	recall
5319 6678	classic show: I ___ ___
1208 5319	___ ___ tuna
8712 8534	there are two houses
6484 8534	we moved forward; we made _____
6484 1020	multiple TV shows might be called?
6665 1332	we're no strangers to love
1020	male sheep (plural)
9789 6665	this can pop a balloon
3601 2278	admiral ackbar
2278 1236	found on some theatre stages
1236	what's behind ___ number one

Chapter 28

Secret sharing

Suppose we are tasked with security at a bank. We set a long combination for the lock on the vault. We want to make sure that none of the three bank's board members can open the vault on her own, but that all three are required to do so. We might break the combination into three portions and give one to each board member. However, this scheme has problems. If we choose a combination that is memorable, it may be possible for two board members to guess the third portion of the combination. And what if we want any two of the board members to be able to open the vault? We could give each person two different portions of the combination in a clever way, but this could quickly get challenging in more complicated schemes.

There are several methods available for distributing a secret to n people such that any k of them can obtain the secret, but such that fewer than k people gain no information at all about the secret. Here is one such method.

1. Convert the secret into a number M.
2. Generate $k-1$ random numbers $a_1, \ldots, a_{k-1}$.
3. Construct the polynomial $f(x) = M + a_1 x + a_2 x^2 + \cdots + a_{k-1} x^{k-1}$.
4. Distribute one of the numbers $f(1), f(2), \ldots, f(n)$ to each of the n people.

Since k points uniquely determine a polynomial of degree $k-1$, any k of the people can determine f and recover the secret $f(0) = M$. However, there are infinitely many polynomials of degree $k-1$ passing through any $k-1$ points, so fewer than k people gain no information about M.

Example 28.1. We want to share the secret $M = 3$ to four people such that any three of them can recover the secret, but no fewer can. In this case, $n = 4$ and $k = 3$. We generate random coefficients and form the polynomial $f(x) = 3 - 2x + x^2$. We distribute the numbers $f(1) = 2$, $f(2) = 3$, $f(3) = 6$, and $f(4) = 11$ as point coordinates to the four people sharing the secret.

Suppose that the first three people want to recover the secret. They know the polynomial is of the form $f(x) = a_0 + a_1 x + a_2 x^2$, but do not know its coefficients a_0, a_1, a_2. They use their values to construct the following equations:

$$\begin{aligned} 2 &= a_0 + a_1 + a_2 \\ 3 &= a_0 + 2a_1 + 4a_2 \\ 6 &= a_0 + 3a_1 + 9a_2 \end{aligned}$$

Algebra gives us the correct coefficients and the secret.

28.1 Exercises

1. Find two second-degree polynomials passing through the points $(1, 2)$ and $(2, 0)$. What implications does this observation have for secret sharing?

2. Explain why the choice of distributed points is arbitrary. That is, why could we distribute any n points (except the secret, of course)?

3. Suppose a secret has been distributed to four people such that any two of them can recover the secret. Two of the coordinates distributed are $f(1) = 4$ and $f(3) = 10$. Determine the secret. Explain geometrically why no one person can recover the secret.

Chapter 29

Homomorphic encryption

Cloud computing is growing in popularity, but with it comes serious security implications. In particular, for any cloud service to perform work on data we store on it, the service must be able to read it. What if you want to keep the data private? One solution is called *homomorphic encryption*.

Let's look at a simple example of what we mean.

Example 29.1. Suppose that Bob is lazy and wants Alice to add two numbers x and y for him. However, he does not want Alice to know what the numbers are. Bob chooses a random number N and adds it to each of x and y. He then hands Alice the numbers $x+N$ and $y+N$. Alice computes the sum $(x+N)+(y+N) = x+y+2N$ and returns it to Bob, who subtracts $2N$ to recover the sum he wants. Of course, since Bob still needs to perform addition and subtraction on his own, he gains very little from this scheme. However, the numbers are kept private from Alice.

While this scheme has some of the properties we are looking for, it is not a homomorphic scheme. Let's define precisely what we mean by a homomorphic scheme.

Definition 29.2. A function f is called *homomorphic with respect to addition* if $f(x+y) = f(x)+f(y)$ for any inputs x and y.

Why is Bob's addition method not homomorphic with respect to addition? Let $f(x) = x+N$. This is the function Bob used to obscure his data before sending it to Alice. Then Alice computes $f(x)+f(y) = x+y+2N$, but if Bob had added the numbers first and then obscured them, he would have computed $f(x+y) = x+y+N$. Since these quantities are not the same, the scheme is not homomorphic.

Are there any schemes that are homomorphic? Yes! Let's define another notion of a homomorphic function.

Definition 29.3. A function f is called *homomorphic with respect to multiplication* if $f(xy) = f(x)f(y)$ for any inputs x and y.

Example 29.4. Consider the process of RSA encryption, where our ciphertext is $M^e \bmod N$ for a message M, key e, and modulus N. Let's define our function $f(x) = x^e \bmod N$. Then

$$\begin{aligned} f(xy) &= (xy)^e \bmod N \\ &= x^e y^e \bmod N \\ &= (x^e \bmod N)(y^e \bmod N) \\ &= f(x)f(y) \end{aligned}$$

and the scheme is homomorphic with respect to multiplication. However, it is not homomorphic with respect to addition.

Are there any schemes that are homomorphic with respect to both addition and multiplication? Such a scheme is called *fully homomorphic*. The first fully homomorphic encryption scheme was devised by Craig

Gentry of IBM in 2005. The mathematics involved are formidable; Gentry uses a jewelry store analogy to help guide the methods.

Why would a homomorphic encryption scheme be of value? Suppose our computer is unable to handle complex computations that we wish to remain a secret. If we give the data to a third-party service with greater computational power unencrypted, we lose the privacy of the information. However, with a homomorphic scheme, we can first encrypt our data, allow the service to perform the computations on the encrypted data using the scheme, and then decrypt the result when we receive it.

To illustrate this, suppose we have pieces of data x and y, and wish to perform a complex computation with result $x \otimes y$, where $\otimes$ is the computationally-intensive operation of interest. It is typically possible to reduce such an operation to a series of additions and multiplications, so a homomorphic scheme is suited to the task. We use the following process:

1. We encrypt the data x and y on our own computer and obtain ciphertexts $C(x)$ and $C(y)$, respectively.
2. We transmit $C(x)$ and $C(y)$ to the third-party service. Since the homomorphic scheme provides privacy and we have kept the encryption key secret, the service cannot determine x or y.
3. The service computes the computationally-intensive operation $C(x) \otimes C(y)$ and transmits this result back to us.
4. Since the homomorphic scheme guarantees that $C(x) \otimes C(y) = C(x \otimes y)$, we can decrypt the result using the key to obtain $x \otimes y$ as desired.

29.1 Exercises

1. Let $f(x) = 3x$. Determine if f is homomorphic with respect to either addition or multiplication.
2. Read the sections of the paper [1] by Craig Gentry that use the jewelry store analogy to explain fully homomorphic encryption. Feel free to skip the sections involving heavy mathematics. Write three questions you have about the material in the paper.
3. Read the paper [2] by Brian Hayes that provides another explanation of homomorphic encryption and applications. Write three questions you have about the material in the paper.

Chapter 30

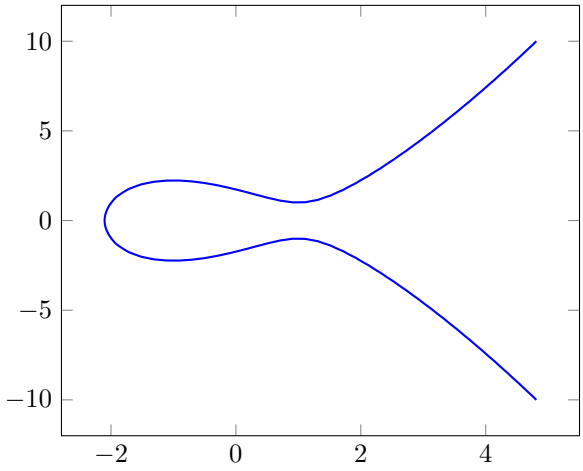
Elliptic curve cryptography

Many modern cryptographic algorithms are based on solving difficult mathematical problems. For example, RSA security comes from the difficulty of factoring large numbers in primes. Diffie-Hellman security comes from the difficulty of solving the discrete logarithm problem. Are other algorithms based on other hard problems? Absolutely! Before we introduce a popular one, we need some mathematical background.

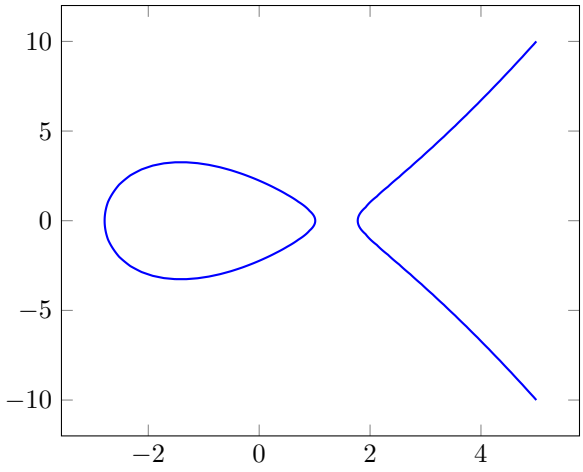
30.1 Elliptic curves

An *elliptic curve* is the set of points (x, y) that satisfy an equation of the form $y^2 = x^3 + ax + b$ for constants a and b, subject to additional restrictions that we will discuss later.

Example 30.1. The curve defined by the equation $y^2 = x^3 - 3x + 3$ is an elliptic curve. This curve is shown below.



Example 30.2. The curve defined by the equation $y^2 = x^3 - 6x + 5$ is an elliptic curve. This curve is shown below.



Notice that not every value of x gives a corresponding value of y that lies on the curve in this example.

It turns out that it is possible to define an addition operation on points that lie on an elliptic curve. It turns out that many modern cryptographic systems rely on such a construction. We will define this addition graphically first, and then determine how to numerically compute sums. It will be left as an exercise to show that the methods are equivalent to each other.

Suppose P and Q are two points on an elliptic curve. We define the sum $P \oplus Q$ as follows:

1. Connect P and Q by a line.
2. Find the point R where this line intersects the curve again.
3. Find the reflected point R' by drawing a vertical line through R and finding where this line intersects the curve again.
4. The sum is $P \oplus Q = R'$.

We note that every point on the curve has a reflected point. This is because replacing y with $-y$ in the equation defining the curve does not change it. That is, the curve is symmetric about the x-axis.

There are a few cases we should consider. First, what happens if we want to find the sum $P \oplus P$? There is no obvious way to connect P to itself with a line! The method we use is to find the line that just "grazes" the curve at P. It is possible to use calculus to define this formally, but we will not do so here. We then continue the same procedure as above.

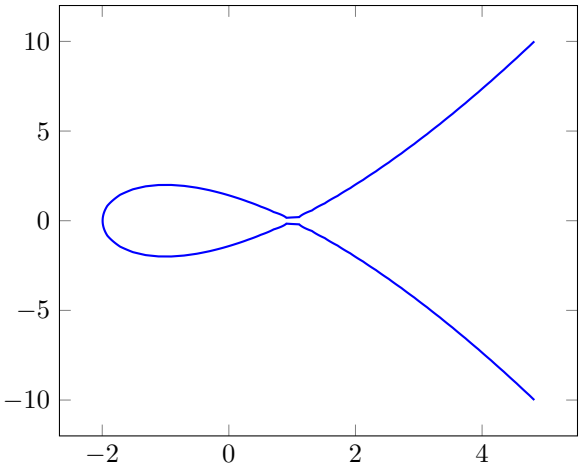
The second case is more subtle. What happens if we want to find the sum $P \oplus P'$, where these points lie on a vertical line? There is no third point on the curve that intersects this line! The solution is to define a special point $\mathcal{O}$ that somehow "lives" at infinity. We can't plot this point since it's just an abstraction, but we define $P \oplus P' = \mathcal{O}$.

The final case will follow from the second. What happens if we want to find the sum $P \oplus \mathcal{O}$? We are somehow connecting P to our special point at infinity. We define this sum using the vertical line passing through P, which reflects to give the point P back again.

For completeness, here is the formal definition of an elliptic curve that we will use.

Definition 30.3. An *elliptic curve* is the set of points satisfying the equation $y^2 = x^3 + ax + b$ for constants a and b, where we require $4a^3 + 27b^2 \neq 0$.

Example 30.4. The curve defined by the equation $y^2 = x^3 - 3x + 2$ is not an elliptic curve since $4(-3)^3 + 27(2^2) = 0$. This curve is shown below.



We have not yet specified how to add points on elliptic curves numerically. Whatever method we use must match with our graphical description of addition. Suppose we have two points $P = (x_p, y_p)$ and $Q = (x_q, y_q)$ on an elliptic curve that do not lie on the same vertical line if they are different (since we already know how to handle that case). We first compute the slope m of the line connecting P and Q, being careful to specify what happens in the case where $P = Q$:

$$m = \begin{cases} \frac{y_q - y_p}{x_q - x_p} & (P \neq Q) \\ \frac{3x_p + a}{2y_p} & (P = Q) \end{cases}$$

The slope in the case where $P = Q$ is computed using calculus; we may take it as given. In either case, the line connecting P and Q can be written easily in point-slope form as $y = m(x - x_p) + y_p$. Remember that our goal is to find the third point on the elliptic curve that intersects this line. To do this, we substitute the equation for the line into the equation governing the elliptic curve. This substitution gives the following:

$$(mx - mx_p + y_p)^2 = x^3 + ax + b$$

Notice that the only unknown is x. This cubic equation has three solutions, two of which we already know to be x_p and x_q. Solving for the third value will give the remaining intersection point x_r, at which point we are almost finished.

We rearrange the above equality by grouping powers of x:

$$x^3 - m^2x^2 + (a - 2m(y_p - mx_p))x + (b - (y_p - mx_p)^2) = 0$$

Since the cubic must factor to give the three intersection points, we have the following:

$$x^3 - m^2x^2 + (a - 2m(y_p - mx_p))x + (b - (y_p - mx_p)^2) = (x - x_p)(x - x_q)(x - x_r)$$

Notice that the coefficient of x^2 on the right-hand side is $-x_p - x_q - x_r$, so matching this to the corresponding term on the left-hand side means that $x_r = m^2 - x_p - x_q$. The equation for the connecting line tells us that $y_r = m(x_r - x_p) + y_p$. However, the final point is the reflection of the point $R = (x_r, y_r)$ across the x-axis. Hence we have the coordinates of the final point R':

$$R' = (x_r, -y_r) = (m^2 - x_p - x_q, -m(x_r - x_p) + y_p)$$

Now that we know how to add points on an elliptic curve, we can define multiples of a point.

Definition 30.5. Let P be a point on an elliptic curve, and let n be a positive integer. We define

$$nP := \underbrace{P \oplus P \oplus \cdots \oplus P}_{n \text{ times}}.$$

30.2 Modular elliptic curves

Since many cryptographic algorithms require that perform modular operations, we can define modular elliptic curves by requiring that the constants a and b, as well as all point coordinates x and y, be integers $\bmod p$, where we require p to be prime for technical reasons. This is the method used in cryptographic algorithms.

How does this affect our addition operation? Practically speaking, we know that we need to perform additions and multiplications using residues. The only tricky operation arises when computing the slope m of the line between the points. Instead of performing division, we instead multiply by the modular inverse of the denominator, as we have done before.

That is, the slope is computed instead as follows:

$$m = \begin{cases} (y_q - y_p)(x_q - x_p)^{-1} \bmod p & (P \neq Q) \\ (3x_p + a)(2y_p)^{-1} \bmod p & (P = Q) \end{cases}$$

30.3 Elliptic discrete logarithm problem

One common elliptic curve algorithm is based on the difficulty of solving a problem much like the standard discrete logarithm problem that we have seen previously. The *elliptic discrete logarithm problem* requires us to solve the equation $Q = nP$, where P and Q are known points, for n. In general, this problem is extremely difficult to solve.

We can use the difficulty of solving the elliptic discrete logarithm problem to define a key exchange method, called *elliptic Diffie-Hellman* key exchange because of its similarity to the standard Diffie-Hellman method. Suppose that Alice and Bob want to use elliptic curves to find a shared secret that their adversary Eve cannot determine efficiently.

1. Alice and Bob determine an elliptic curve and a point P on the curve. If Eve is monitoring their communications, she also knows these.
2. Alice chooses a secret integer n_A and sends the point $Q_A := n_A P$ to Bob. Eve also knows this point.
3. Bob chooses a secret integer n_B and sends the point $Q_B := n_B P$ to Alice. Eve also knows this point.
4. Alice computes the point $n_A Q_B = n_A(n_B P) = (n_A n_B)P$.
5. Bob computes the point $n_B Q_A = n_B(n_A P) = (n_B n_A)P$.
6. Alice and Bob use this computed point as their shared secret.

Why are Alice and Bob guaranteeed to obtain the same point? Since n_A and n_B are integers, we have $n_A n_B = n_B n_A$, so the points $(n_A n_B)P$ and $(n_B n_A)P$ are equal. For Eve to determine this point, she would need to solve the elliptic discrete logarithm problem to determine either n_A or n_B, and use the value to compute the shared point.

There are methods for efficiently computing repeated elliptic curve point addition, but we will not discuss them here. The next example will provide an indication of how elliptic Diffie-Hellman key exchange works.

Example 30.6. Suppose that Alice and Bob want to use elliptic Diffie-Hellman key exchange to establish a shared secret, since their communication channel is being monitored by Eve. They choose the elliptic curve $y^2 = x^3 + 324x + 1287$ and require that all computations be done $\bmod 3851$. They also choose the common point $P = (920, 303)$. Alice chooses the secret integer $n_A = 1194$ and sends the point $Q_A = 1194P = (2067, 2178)$ to Bob. Bob chooses the secret integer $n_B = 1759$ and sends the point $Q_B = 1759P = (3684, 3125)$ to Alice. Alice computes $n_A Q_B = 1194(2684, 3125) = (3347, 1242)$, while Bob computes $n_B Q_A = 1759(2067, 2178) = (3347, 1242)$. This common point is their shared secret, which Eve cannot determine efficiently.

It is possible to use elliptic curves to define other cryptographic algorithms.

30.4 Exercises

1. Let P and Q be points on an elliptic curve. The following exercises show how elliptic addition acts like standard addition in some ways, and can be used to establish a commutative group structure. For this exercise, we will not show associativity formally since it is tedious.

   (a) Show that elliptic curve point addition is a closed operation. That is, show that the sum of any two points on an elliptic curve is always another point on the curve.

   (b) Explain why $P \oplus \mathcal{O} = P$. That is, show that the point $\mathcal{O}$ has the properties of the identity element.

   (c) Explain why for any non-identity point P on an elliptic curve, its inverse is the point P'.

   (d) Explain why $P \oplus Q = Q \oplus P$.

Chapter 31

Message authentication codes

We have seen how to apply concepts from public-key cryptography to verify the authenticity and integrity of messages. That is, we used hash functions and digital signatures to ensure that a message came from a given sender, and that the message was not tampered with by any attacker.

It is natural to ask if similar techniques exist without requiring the use of asymmetric keys. The answer is yes; we will introduce a few definitions to help us examine how.

Definition 31.1. A *message authentication code* (often abbreviated MAC) is an algorithm that takes a message and key and generates a piece of information called a *tag* that can be used to verify the integrity and authenticity of the message.

Notice that this definition is quite vague. What are examples of MAC algorithms? How do they work? Let's first examine a general approach of how a MAC algorithm would be used in practice. Suppose that Alice wants to send a message to Bob, who wants to ensure that the message came from Alice and was not tampered with by Eve in transit. Suppose the message itself is public; that is, Alice and Bob have no reason to encrypt the message.

1. Alice and Bob agree beforehand on a secret key.
2. Alice takes her message and the secret key, and uses a MAC algorithm to generate a tag.
3. Alice sends the message and tag (but not the secret key) to Bob. It is assumed that Eve has access to both of these.
4. Bob takes the received message and the secret key, and uses the MAC algorithm to generate his own tag.
5. If the two tags agree, Bob is assured that the message came from Alice and was not tampered with by Eve.

The security of this system hinges on a few key points.

- Alice and Bob must be able to exchange a secret key beforehand, using a communication channel that Eve does not have access to.
- Eve must not be able to generate her own message and tag in a way that could convince Bob that Alice sent the message.
- Eve must not be able to tamper with Alice's original message in a way that results in the same tag; otherwise, Bob will not know that the message he received is the same message that Alice sent.

This seems like a tall order. Fortunately, we already have the tools we need to generate certain types of MAC algorithms. There are several ways to do so.

31.1 Block cipher message authentication codes

Suppose we have a block cipher F_k (using secret key k) that encrypts messages one block at a time. We can define a MAC algorithm that, for any single-block message m, generates the tag $F_k(m)$. This MAC algorithm satisfies the requirements we listed above. Eve cannot generate her own tag since she does not know the secret key k used in F_k, and it is unlikely she can alter the message such that the block cipher encrypts her altered message to the same tag.

However, this scheme only applies to message that have the same length as the block size of our cipher, which is typically not very large. We need a way to alter the algorithm to handle larger messages. Suppose we want to encrypt messages of l blocks, where l is a fixed number. A type of MAC called a *cipher block chaining message authentication code* or CBC-MAC operates by encrypting each block in the message $m = m_1 m_2 \cdots m_l$ with the block cipher F_k, taking the tag and XORing it with the next block, encrypting this new block with F_k, and continuing until we have the last tag t. We then send the original message and this last tag. This scheme is demonstrated in Figure 31.1.

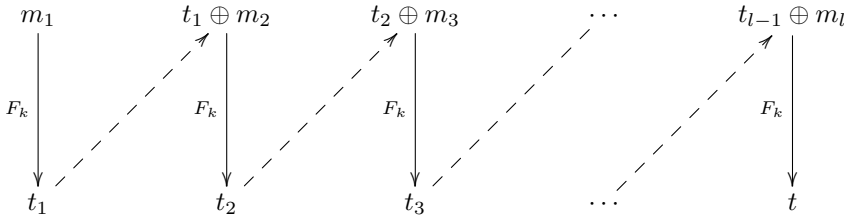


Figure 31.1: CBC-MAC algorithm

It should be noted that this algorithm is only secure when the length of the message is fixed to l blocks. In the exercises, we investigate why allowing messages of any number of blocks renders the CBC-MAC technique insecure. There are other techniques based on block ciphers that allow for messages of arbitrary length in a secure way.

31.2 Hash-based message authentication codes

It is possible to use hash functions to define MAC algorithms, but surprisingly tricky to do so in a secure way. Such a method is called a *hash-based message authentication code* or HMAC algorithm.

To illustrate why using hashes to generate MAC algorithms is difficult to do securely, consider the following method. Suppose we take a hash function H constructed from a Merkle-Damgård transform. To generate our tag, we concatenate a secret key k and message m (denoted by $k||m$) and compute $H(k||m)$. While this seems like a reasonable approach, it turns out that if the message and length of the key are known (but not the key itself), it is possible to generate other valid tags for new messages, rendering the entire method insecure. In particular, MD5 and SHA-1 are vulnerable to this attack, known as a *length-extension attack*. The online photo service Flickr was vulnerable to precisely this attack, which allowed attackers to execute commands on unsuspecting users' accounts until the vulnerability was discovered. We detail how to execute a length-extension attack in the exercises.

One way to construct a better HMAC is to both prepend and append the key to the message; that is, the tag for a message m with key k is $H(k||m||k)$. It has been suggested that even this approach is weak, so an even better scheme is used in practice; it outputs the tag $H(k||H(k||m))$.

Example 31.2. Many two-factor authentication systems, like those used by Google and Facebook, use HMAC techniques to generate six-digit codes used when authenticating with online services. To use these systems, a user enters her credentials as normal, but also enters a six-digit code generated by a mobile application on her phone or tablet. The six-digit code changes every thirty seconds. If the user enters an incorrect code, the system will not authenticate her.

To set up two-factor authentication, the user is given a random-looking string of characters that she enters into the mobile application when she installs it. This is the secret key k used in an HMAC algorithm on the mobile application. To generate the code, the application uses the current time (rounded to thirty seconds) as the HMAC message. The algorithm generates a tag that is converted into a six-digit code for convenience.

This two-factor authentication system is secure since an attacker is unable to generate his own HMAC tags without knowing the secret key used in the algorithm. The scheme is not vulnerable to replay attacks since the code changes every thirty seconds. In practice, the authenticating service will accept codes within a certain time range (typically about five minutes) in case the clock on the user's mobile device is not completely accurate. However, replay attacks outside of this small time range are not possible.

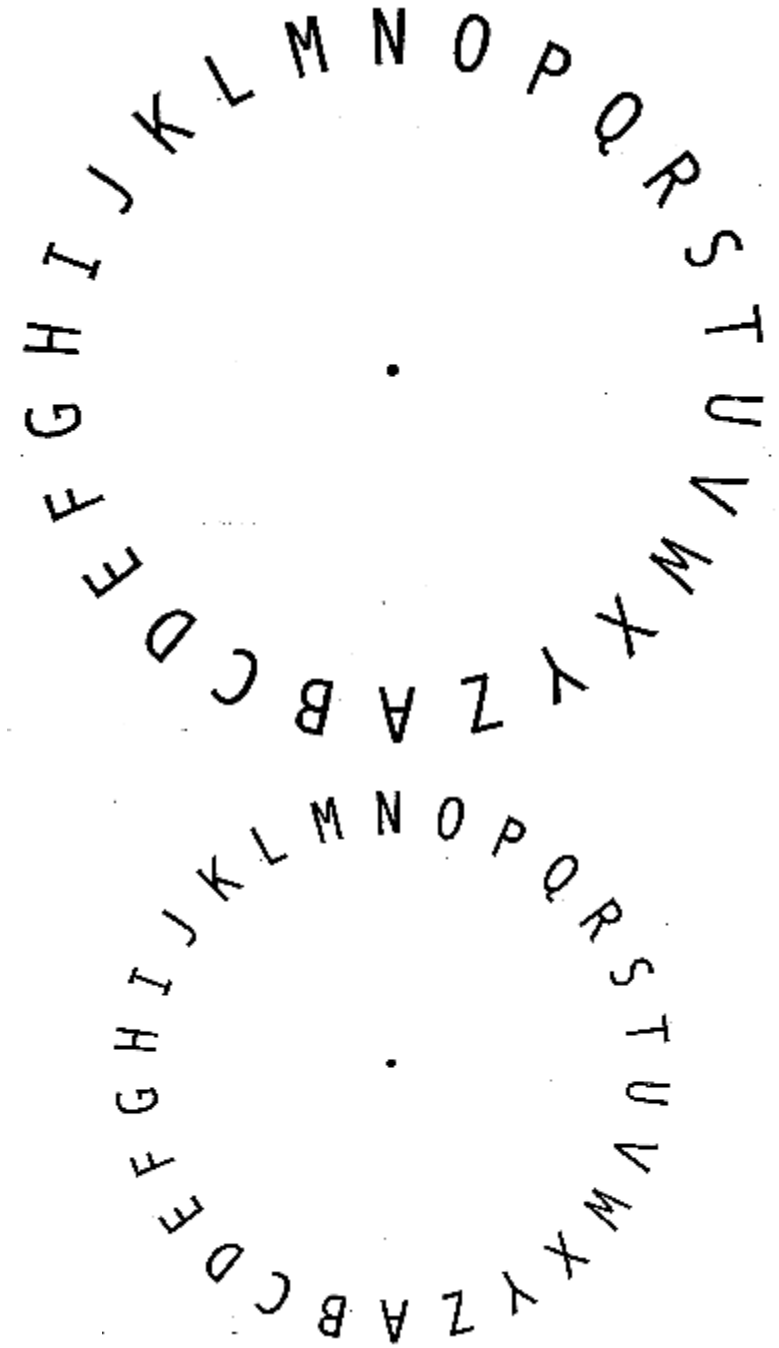
31.3 Exercises

1. Suppose an adversary obtains messages m and m', along with their CBC-MAC tags t and t'. Suppose the adversary takes m, appends the block $m'_1 \oplus t$ to it (where m'_1 is the first block of m'), and then appends the remaining blocks of m'. Show that this new message has the tag t'. Why does this imply that allowing variable-length messages in a CBC-MAC is insecure?

2. Suppose we generate an HMAC-like tag using a hash function H constructed from a Merkle-Damgård transform, as described above. That is, for a secret key k and message m, the tag is $H(k||m)$. Suppose we receive a message and tag such that we know the length of the secret key but not the key itself. We can use this information to determine the zero-padding p that was used when computing the tag; that is, $H(k||m) = H(k||m||p)$. Show that we can compute a valid tag for the message $k||m||p||m'$, where m' is any message of our choice. Explain why this length-extension attack renders this HMAC-like algorithm insecure.

Appendix A

Caesar wheel



Appendix B

Frequency table

This table gives letter frequency percentages for long English texts.

A	B	C	D	E	F	G	H	I	J	K	L	M
8.2	1.5	2.8	4.3	12.7	2.2	2.0	6.1	7.0	0.2	0.8	4.0	2.4

N	O	P	Q	R	S	T	U	V	W	X	Y	Z
6.7	7.5	1.9	0.1	6.0	6.3	9.1	2.8	1.0	2.4	0.2	2.0	0.1

Appendix C

Vigenère square

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
0	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
1	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
2	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
3	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
4	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
5	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
6	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
7	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
8	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
9	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
10	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
11	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
12	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
13	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
14	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
15	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
16	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
17	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
18	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
19	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
20	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
21	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
22	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
23	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
24	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
25	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

Appendix D

Frequency tables for Vigenère exercises

Letter	1	2	3	4	5
A	1.9	1.9	5.7	1.9	0.0
B	3.8	3.8	13.2	7.5	0.0
C	5.7	0.0	11.3	1.9	1.9
D	7.5	0.0	0.0	0.0	0.0
E	3.8	0.0	1.9	11.3	5.7
F	0.0	0.0	1.9	11.3	1.9
G	3.8	0.0	9.4	13.2	3.8
H	3.8	7.5	11.3	5.7	3.8
I	17.0	1.9	5.7	0.0	9.4
J	1.9	3.8	3.8	0.0	1.9
K	0.0	3.8	1.9	0.0	0.0
L	1.9	18.9	0.0	3.8	5.7
M	0.0	5.7	0.0	0.0	9.4
N	0.0	0.0	0.0	1.9	0.0
O	0.0	3.8	1.9	0.0	0.0
P	9.4	9.4	1.9	0.0	3.8
Q	1.9	0.0	5.7	1.9	1.9
R	0.0	0.0	5.7	20.8	5.7
S	5.7	5.7	5.7	0.0	5.7
T	18.9	0.0	3.8	1.9	1.9
U	5.7	5.7	0.0	3.8	0.0
V	1.9	15.1	0.0	9.4	11.3
W	3.8	3.8	5.7	0.0	13.2
X	3.8	0.0	1.9	0.0	11.3
Y	0.0	9.4	0.0	1.9	1.9
Z	0.0	1.9	3.8	1.9	0.0

Table D.1: Frequency table for Vigenère exercise 7

Letter	1	2	3	4
A	6.4	4.3	0.0	4.3
B	0.0	10.6	2.1	0.0
C	10.6	6.4	0.0	4.3
D	2.1	8.5	12.8	0.0
E	2.1	0.0	0.0	6.4
F	6.4	8.5	2.1	0.0
G	10.6	4.3	6.4	2.1
H	2.1	2.1	6.4	10.6
I	6.4	4.3	2.1	17.0
J	4.3	0.0	2.1	0.0
K	4.3	2.1	0.0	4.3
L	0.0	0.0	8.5	2.1
M	2.1	2.1	0.0	0.0
N	4.3	0.0	0.0	0.0
O	0.0	10.6	6.4	0.0
P	2.1	0.0	4.3	4.3
Q	8.5	2.1	14.9	6.4
R	8.5	2.1	10.6	12.8
S	0.0	12.8	0.0	8.5
T	4.3	2.1	2.1	4.3
U	2.1	2.1	12.8	0.0
V	6.4	4.3	2.1	4.3
W	6.4	12.8	2.1	0.0
X	0.0	0.0	0.0	6.4
Y	2.1	0.0	4.3	2.1
Z	0.0	0.0	0.0	0.0

Table D.2: Frequency table for Vigenère exercise 8

Letter	1	2	3
A	0.9	4.5	8.2
B	5.5	0.0	0.9
C	7.3	0.9	4.5
D	0.0	3.6	3.6
E	0.0	7.3	16.4
F	2.7	8.2	0.9
G	0.0	6.4	0.0
H	9.1	3.6	12.7
I	8.2	0.9	3.6
J	0.9	2.7	0.0
K	0.9	5.5	0.0
L	7.3	13.6	2.7
M	9.1	0.9	0.9
N	8.2	0.0	4.5
O	4.5	3.6	10.0
P	0.9	0.0	3.6
Q	0.9	0.0	0.0
R	0.0	0.0	2.7
S	0.9	6.4	5.5
T	0.0	0.0	11.8
U	9.1	4.5	2.7
V	0.9	1.8	1.8
W	2.7	15.5	1.8
X	4.5	2.7	0.0
Y	13.6	0.9	0.9
Z	1.8	6.4	0.0

Table D.3: Frequency table for Vigenère exercise 9

Letter	1	2	3	4	5
A	0.0	7.7	0.0	0.0	10.3
B	0.0	0.0	0.0	5.1	7.7
C	0.0	0.0	7.7	7.7	0.0
D	0.0	5.1	5.1	0.0	0.0
E	10.3	10.3	15.4	0.0	5.1
F	7.7	0.0	12.8	10.3	10.3
G	2.6	7.7	2.6	0.0	7.7
H	5.1	2.6	0.0	12.8	5.1
I	2.6	5.1	5.1	10.3	2.6
J	0.0	0.0	2.6	0.0	2.6
K	2.6	0.0	5.1	2.6	0.0
L	2.6	2.6	2.6	0.0	2.6
M	5.1	5.1	0.0	2.6	0.0
N	15.4	7.7	5.1	0.0	7.7
O	20.5	7.7	0.0	5.1	0.0
P	0.0	0.0	5.1	0.0	2.6
Q	0.0	0.0	0.0	0.0	2.6
R	0.0	2.6	10.3	7.7	10.3
S	2.6	5.1	0.0	5.1	0.0
T	12.8	5.1	0.0	2.6	5.1
U	5.1	10.3	0.0	5.1	2.6
V	2.6	5.1	15.4	0.0	5.1
W	5.1	2.6	0.0	12.8	2.6
X	0.0	0.0	2.6	0.0	5.1
Y	0.0	10.3	5.1	0.0	0.0
Z	0.0	0.0	0.0	10.3	2.6

Table D.4: Frequency table for Vigenère exercise 10

Appendix E

Modular inverses

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$n^{-1} \bmod 29$														
n	15	16	17	18	19	20	21	22	23	24	25	26	27	28
$n^{-1} \bmod 29$														

Appendix F

Enigma simulator

The following three pages contain a paper Enigma simulator. To assemble it, follow these steps:

1. On the first page, cut the three dashed lines near the top and bottom of the page, as well as the two dashed lines at the left of the page.
2. On the second and third pages, cut out the three alphabet strips and four plugboards on each page along the dotted lines.
3. Match the alphabet strips by number. For each set of strips, cover the tab at the top of the strip from the third page (marked "link") with the bottom of the corresponding strip from the second page; apply clear tape. When finished, each strip will have a tab at the top and bottom, with a shaded line in the middle, to form an uninterrupted set of letters.

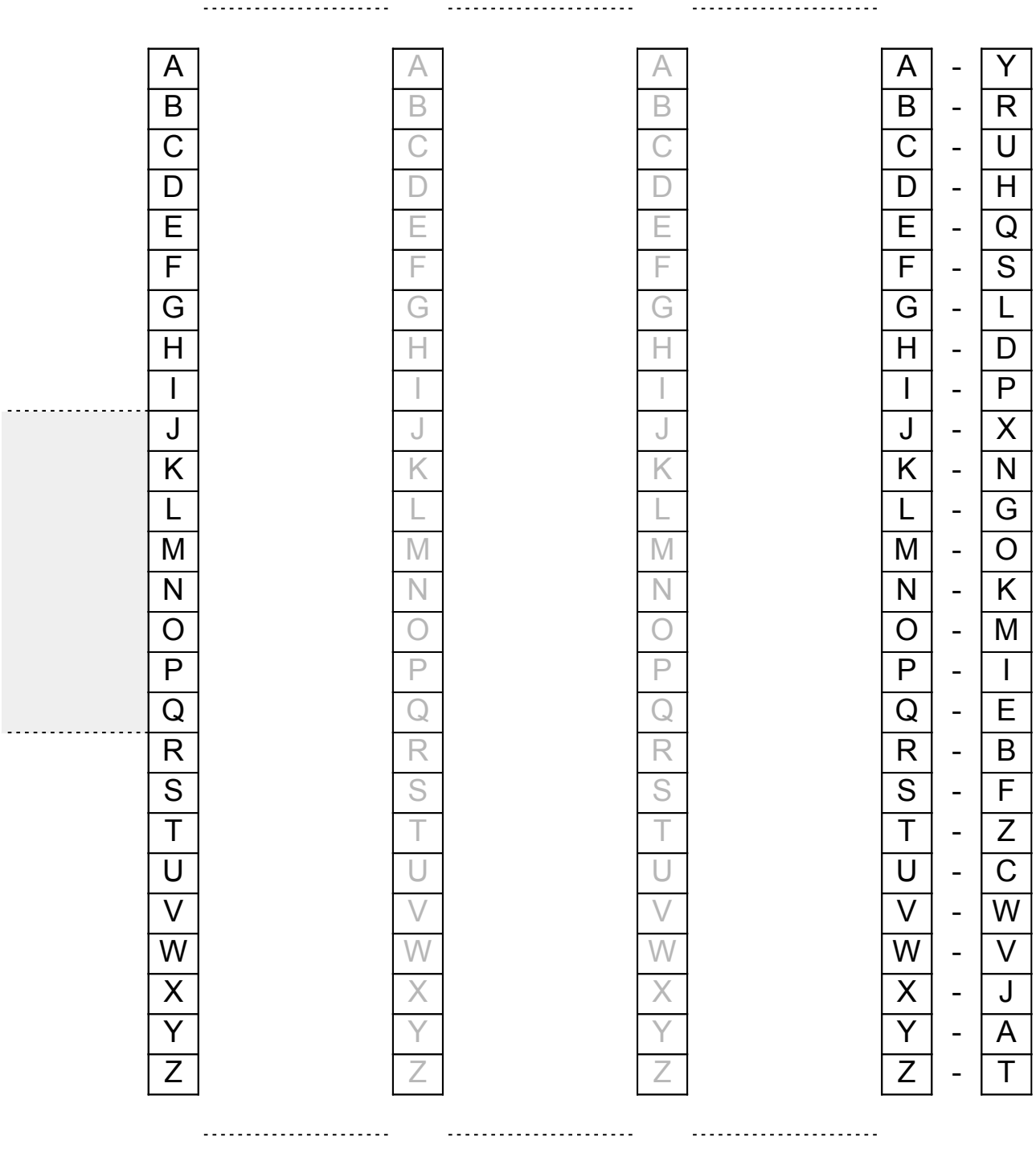
To set the simulator, follow these steps:

1. Choose a plugboard. Insert the plugboard into the two slots at the left of the simulator.
2. Choose a rotor order. Insert the three rotor strips into the corresponding slots.
3. Choose an initial rotor orientation. For each rotor, move the shaded line so it lines up with the desired letter to its left.

To encrypt or decrypt a message after the simulator is set, follow these steps:

1. If the letter appears on the plugboard, swap it with the listed letter.
2. Find this letter in the list to the left of the first rotor. Follow it into the first rotor and find the matching letter on the right of the rotor.
3. Exit the first rotor and enter directly into the second. Find the matching letter on the right of the rotor.
4. Exit the second rotor and enter directly into the third. Find the matching letter on the right of the rotor.
5. Exit the third rotor to the reflector. Determine the reflected letter.
6. Go through the rotors again, this time in reverse order.
7. Apply the plugboard if needed after the first rotor.
8. Move the first rotor down one letter. If the gray strip was already at the bottom letter, move it to the top and move the second rotor down one letter. If its gray strip was already at the bottom letter, move it to the top and move the third rotor down one letter. If its gray strip was already at the bottom letter, move it to the top.
9. Repeat these steps for each letter of the message.

Enigma Simulator



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

1	2	3

E - K	E - W
N - J	S - N
I - P	T - U
G - Y	J - Z
M - O	I - P
A - W	B - M
1	2

A - H	H - Z
L - U	V - E
C - J	R - J
K - X	U - Q
W - E	A - X
R - D	F - I
3	4

1		2		3						
K		B	J	B	D	B	M - V	Q - D		
M		C	D	C	F	C	G - Y	N - B		
F		D	K	D	H	D	E - C	F - X		
L		E	S	E	J	E	K - N	Z - Y		
G		F	I	F	L	F	H - R	J - S		
D		G	R	G	C	G	L - P	W - A		
Q		H	U	H	P	H				
V		I	X	I	R	I	5	6		
Z		J	B	J	T	J				
N		K	L	K	X	K				
T		L	H	L	V	L				
O		M	W	M	Z	M				
W		N	T	N	N	N				
Y		O	M	O	Y	O				
H		P	C	P	E	P	I - K	Q - K		
X		Q	Q	Q	I	Q	N - U	C - Z		
U		R	G	R	W	R	X - B	A - F		
S		S	Z	S	G	S	E - F	R - O		
P		T	N	T	A	T	L - Y	X - P		
A		U	P	U	K	U	C - W	M - Y		
I		V	Y	V	M	V				
B		W	F	W	U	W	7	8		
R		X	V	X	S	X				
C		Y	O	Y	Q	Y				
J		Z	E	Z	O	Z				
E		**A**	**A**	**A**	**B**	**A**				
link		link		link						

Appendix G

Letter encoding

Letter	$a \bmod 29$	Binary	$a^{-1} \bmod 29$
A	0	00000	–
B	1	00001	1
C	2	00010	15
D	3	00011	10
E	4	00100	22
F	5	00101	6
G	6	00110	5
H	7	00111	25
I	8	01000	11
J	9	01001	13
K	10	01010	3
L	11	01011	8
M	12	01100	17
N	13	01101	9
O	14	01110	27
P	15	01111	2
Q	16	10000	20
R	17	10001	12
S	18	10010	21
T	19	10011	26
U	20	10100	16
V	21	10101	18
W	22	10110	4
X	23	10111	24
Y	24	11000	23
Z	25	11001	7
_	26	11010	19
,	27	11011	14
.	28	11100	28
?	29	11101	–
/	30	11110	–
’	31	11111	–

Appendix H

List of activities

H.1 Random number generation

The goal of this activity is to examine the difference between random number generation and pseudorandom number generation. Split into two teams. Each team first writes down a "random" list of 20 Hs and Ts, in any order, on a sheet of paper. The team then flips a coin 20 times and writes down the resulting Hs and Ts (for heads and tails, respectively) on an indentical sheet of paper. It may be useful to further split the team into groups, each armed with a coin, to complete a portion of the flips.

When both teams have their two lists, they exchange them without knowing which list is from the coin flips and which is from the made-up flips. The teams must try to guess which is which. The teams should list what qualities of each list led them to their choice.

H.2 Combinatorics of the SET card game

The goal of this activity is to understand the application of combinatorics to the SET card game. Split into groups of about five people each. Each groups needs a copy of the SET card game. Play a few rounds of the game to get a sense of the patterns that you find in the cards. Then answer the following questions:

1. The SET card game includes cards with every possible combination of number, shape, shading, and color. How many cards are there? How do you obtain this number?

2. Suppose you draw two cards at random from the SET deck. Is it always possible to find a third card to complete a SET? How many such SETs exist for two given cards?

3. Suppose you draw one card at random from the SET deck. Is it always possible to find two additional cards to complete a SET? How many such SETs exist for a given card?

4. How many cards do you think can be dealt such that no SET appears among them? This is a very difficult problem, so make an educated estimate.

H.3 Tower of Hanoi

The goal of this activity is to understand principles of counting and recursion. In the Tower of Hanoi puzzle, there are three pegs in a row. On the left peg are n disks of decreasing size arranged in a pyramid; that is, the disks get smaller in size from bottom to top. The goal of the puzzle is to move all disks to the right peg, one at a time. A legal move consists of moving the top disk from any peg to any other peg, as long as no disk ever sits on top of a smaller one. Let h_n denote the fewest number of moves required to solve the Tower of Hanoi problem with n disks.

1. Find h_1, h_2, h_3, and h_4.

2. Develop a recursive formula that establishes h_n in terms of h_{n-1}; that is, if you know the fewest number of moves to solve the $(n-1)$-disk Tower of Hanoi problem, can you use this to figure out the fewest number of moves to solve the n-disk problem? It is not necessary to actually determine a formula for h_{n-1} in this case!

3. Hypothesize an explicit formula for h_n that is not recursive.

H.4 Measuring hash functions

One common statistical requirement for a good hash function is that a change of a single bit of the input changes about half of the bits in the hash. This activity investigates whether modern hash functions obey this principle.

We will check to see if changing a single input bit to an MD5 hash results in a change of about half of the output bits. We will use the following:

	MD5 hexadecimal hash
Moors	`1CA29DFB8B94B8C082DAA4BAA2BC8B03`
Moops	`4BB03083EF11974A0CE92AD241C2F455`

1. We will examine only the last 32 bits of each hash. For each hash, write the hexadecimal digits we need to use to obtain these 32 bits.

2. Write each of these smaller hexadecimal numbers in binary.

3. In how many bits do these numbers differ? Is it close to half?

4. What are some reasons why the two numbers might not differ in exactly half of the bits?

We will investigate the same two words using the SHA-1 hash function.

	SHA-1 hexadecimal hash
Moors	`4B02EE525EDCF52B16F20C88A77BC8A0FAD3EAF1`
Moops	`669FC0C24C6061AC8EBF4A3C60AFC130E82296C4`

1. We will examine only the last 32 bits of each hash. For each hash, write the hexadecimal digits we need to use to obtain these 32 bits.

2. Write each of these smaller hexadecimal numbers in binary.

3. In how many bits do these numbers differ? Is it close to half?

We will investigate the same two words using the SHA-2 hash function.

	SHA-2 hexadecimal hash
Moors	`3C75C1374380D709E46B1E5CE82DFCDDD7F31D91DFD47AE42874A3585F0CF7A7`
Moops	`97756326C7366D99D7AD18446FE1ABF468A5B14D11FEC5A29AE1211E5DE6E34E`

1. We will examine only the last 32 bits of each hash. For each hash, write the hexadecimal digits we need to use to obtain these 32 bits.

2. Write each of these smaller hexadecimal numbers in binary.

3. In how many bits do these numbers differ? Is it close to half?

We will investigate the same two words using the SHA-3 hash function.

	SHA-3 hexadecimal hash
Moors	`E8B46458C7DD6771CAECA7C4B024C654AFFD7096B2C281702D2B5C231BD87A0D`
Moops	`429246D7D0B91819401DF7AEC1B74C75D94F7E7430BBE0ACD4077C1F23F2A322`

1. We will examine only the last 32 bits of each hash. For each hash, write the hexadecimal digits we need to use to obtain these 32 bits.
2. Write each of these smaller hexadecimal numbers in binary.
3. In how many bits do these numbers differ? Is it close to half?

H.5 Secret sharing game

The purpose of this activity is to learn to use the mathematical tools needed in polynomial secret sharing. Scrooge McDuck has locked a great treasure in his vault with a six-digit combination. Because he does not want any one person (or duck) to have any of the digits, he has used a polynomial scheme to distribute the digits.

H.5.1 Equipment

A total of 27 cards in six colors. Each color is labeled with a polynomial degree, and each card has an ordered pair:

- Red, $y = x^2 - 2x + 3$: $(1, 2), (-3, 18), (4, 11), (-2, 11), (-1, 6)$
- Orange, $y = 8x - 4$: $(3, 20), (-2, -20), (1, 4), (9, 68)$
- Yellow, $y = -2x^2 - 2$: $(1, -4), (2, -10), (-1, -4), (-2, -10), (3, -20)$
- Green, $y = -3x + 9$: $(7, -12), (-3, 18), (10, -21), (3, 0)$
- Blue, $y = 2x^2 - 6x + 4$: $(1, 0), (2, 0), (-1, 12), (-2, 24), (3, 4)$
- Purple, $y = 15$: $(19, 15), (-4, 15), (12, 15), (-1, 15)$

H.5.2 Rules

- Divide into groups of two. If there are not enough cards to distribute evenly, remove some cards as needed. There must be at least three red, yellow, and blue cards; there must be at least two orange, green, and purple cards.
- Shuffle the deck and deal all the cards.
- On group A's turn, they can either trade information or guess a digit.
- If group A chooses to trade information, they ask any particular group B if group B has a specific color. If group B does, they choose one of these cards to give to group A; at the same time, group A chooses any card to give to group B and the turn ends. If group B does not have any cards of the given color, the turn ends.
- If instead group A chooses to guess a digit of the combination, they write down the color and their guess for the instructor in secret. If the guess is correct, the turn ends. If the guess is incorrect, the turn ends and group A loses their next turn.
- Play continues to the next group clockwise.
- The game ends when any group guesses all six digits in the combination.

Appendix I

Research projects

Here are several prompts for research.

1. For either the United States, India, Germany, or the United Kingdom, write an essay on the history of public policy restricting encryption techniques in that country. Be sure to include an analysis of current policy.

2. Some argue that because encryption technologies can be used by criminals for illegal purposes, they should be regulated by governments. Others argue that principles of free speech require the free use of such technologies. Analyze both arguments and reach your own conclusion about which argument, if either, is correct. Be sure to justify any claims that you make in your discussion.

3. Recent news about information leaked to the press by former United States National Security Agency (NSA) contractor Edward Snowden details NSA efforts to persuade the security firm RSA to include a weakened algorithm in its suite of security products. Research this story and report on your findings.

4. A massive shift in cryptographic technology occurred with the introduction of public-key cryptography. Investigate recent cryptographic developments and speculate about what the next such shift might be.

5. Services like Dropbox, Microsoft SkyDrive and Google Drive allow users to store files on servers owned by those companies. Users' web connections to those services are typically encrypted, but the files themselves are not. Research and discuss the benefits and drawbacks of this arrangement. Suggest improvements that could solve some of the associated problems.

6. The encrypted email service Lavabit has been in the news recently because of United States National Security Agency (NSA) efforts to obtain encryption key information from the company. Research this story and report on your findings. Discuss the possible implications of the actions taken by the NSA and Lavabit.

7. Originally, email messages were sent from the sender to the recipient unencrypted. Efforts have since been taken to secure the contents of email messages in transit. Research the history of email security technologies and report your findings. Be sure to discuss the current state of email security and comment on recent and ongoing developments.

8. It is not uncommon for researchers to discover flaws in cryptographic algorithms or technologies that render them unsafe to use. Research an example of this and report your findings. Be sure to include a discussion of the algorithm or technology, the flaws that were found, and the impact of the announcement.

9. Investigate the history of hash collisions, focusing particularly on the MD5 hash. Report your findings.

10. Because companies and industries often invest heavily in particular cryptographic technologies, they are sometimes slow to update when better technologies are developed. Find an example of this and report your findings.

References

[1] Craig Gentry. Computing arbitrary functions of encrypted data. *Communications of the ACM*, 53(3):97–105, 2010.

[2] Brian Hayes. Alice and Bob in cipherspace. *American Scientist*, 100:362–367, 2012.

[3] Jeffrey Hoffstein, Jill Pipher, and J.H. Silverman. *An Introduction to Mathematical Cryptography.* Springer Publishing Company, Incorporated, 1st edition, 2008.

[4] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography.* Chapman & Hall/CRC, 1st edition, 2008.

[5] Bruce Schneier. *Applied Cryptography (2nd Ed.): Protocols, Algorithms, and Source Code in C.* John Wiley & Sons, Inc., New York, NY, USA, 2nd edition, 1995.

[6] Bruce Schneier. *Schneier on Security.* Wiley, 1st edition, 2008.

[7] Simon Singh. *The Code Book: The Science of Secrecy from Ancient Egypt to Quantum Cryptography.* Anchor Books, New York, NY, USA, 1st edition, 1999.