Notes on cryptography

Peter J. Cameron
School of Mathematical Sciences
Queen Mary, University of London
Mile End Road
London E1 4NS
UK
p.j.cameron@qmul.ac.uk

Contents

Preface

These notes are associated with the course MAS335, *Cryptography*, given at Queen Mary, University of London, in the autumn semester of 2002. The notes are much improved from my original drafts as a result of comments from the students on the course.

The syllabus for the course reads:

1. History and basic concepts (Substitution and other traditional ciphers; Plaintext, ciphertext, key; Statistical attack on ciphers).
2. One-time pad and stream ciphers (Shannon's Theorem; One-time pad; Simulating a one-time pad; stream ciphers, shift registers).
3. Public-key cryptography (Basic principles (including brief discussion of complexity issues); Knapsack cipher; RSA cipher; Digital signatures).

Optional topics which may be included: secret sharing, quantum cryptography, the Enigma cipher, for example.

Peter J. Cameron
November 27, 2003

Chapter 1

Basic ideas

1.1 Introduction

Cryptography refers to the art of protecting transmitted information from unauthorised interception or tampering. The other side of the coin, *cryptanalysis*, is the art of breaking such secret ciphers and reading the information, or perhaps replacing it with different information. Sometimes the term *cryptology* is used to include both of these aspects. In these notes I will use the term *cryptography* exclusively.

Cryptography is closely related to another part of communication theory, namely *coding theory*. This involves translating information of any kind (text, scientific data, pictures, sound, and so on) into a standard form for transmission, and protecting this information against distortion by random noise. There is a big difference, though, between interference by random noise, and interference by a purposeful enemy, and the techniques used are quite different.

The need for both coding theory and cryptography has been recognised for a long time. Here, from "The Tale of Lludd and Llevelys" in *The Mabinogion* (a collection of ancient Welsh stories), is a tale that illustrates both subjects.

> When Lludd told his brother the purpose of his errand Llevelys said that he already knew why Lludd had come. Then they sought some different way to discuss the problem, so that the wind would not carry it off and the Corannyeid learn of their conversation. Llevelys ordered a long horn of bronze to be made, and they spoke through that, but whatever one said to the other came out as hateful and contrary. When Llevelys perceived there was a devil frustrating them

> and causing trouble he ordered wine to be poured through the horn to wash it out, and the power of the wine drove the devil out.

Here the horn is a cryptographic device, preventing the message from being intercepted by the enemy (the Corannyeid); this is an example of a *secure channel*, which we will discuss later. Pouring wine down the horn is a bizarre form of error-correction.

1.2 Steganography and cryptography

There are two principal ways to keep a message out of the enemy's hands:

- You can conceal the message and hope that the enemy can't find it: this is known as *steganography*.
- You can scramble the message, and hope that (assuming that it is intercepted) the enemy is unable to unscramble it: this is what is properly known as *cryptography*.

We are mainly concerned with cryptography; but here are a few of the many methods of steganography that have been used or proposed.

- Herodotus relates that one Histauaeus shaved the head of his messenger, wrote the message on his scalp, and waited for the hair to regrow. On reaching his destination, the messenger shaved his head again and the recipient, Aristogoras, read the message. Not to be recommended if you are in a hurry!
- Invisible ink comes into this category; the recipient develops the message by applying heat or chemicals to it.
- A message can be concealed in a much longer, innocent-looking piece of text; the long text is composed so that a subsequence of the letters (chosen by some rule known to the recipient) forms the message. For example, taking every fifth letter of

  ```
  The prepared letters bring news of amounts
  ```

  gives the message "Retreat".
- The message can be photographed and reduced to a tiny speck called a *microdot*, which can be concealed in a full stop in an ordinary letter.

- A recent proposal uses the fact that a molecule of DNA (the genetic material in all living things) can be regarded as a very long word in an alphabet of four letters A, C, G, T (the bases adenine, cytosine, guanine and thymine). Now that the technology exists to modify DNA very freely, it is possible to encode the message in this four-letter alphabet and insert this sequence into a DNA molecule. A small amount of DNA can then be concealed in a letter, in the same way as a microdot. (This method may or may not have been used.)

Of course, steganography can be combined with cryptography: the message can be scrambled and then hidden, for extra security.

1.3 Some terms defined

Figure 1.1 shows the general scheme of cryptography. Traditionally, the two parties who want to communicate are called Alice and Bob, and the eavesdropper who is trying to read their message is Eve. Alice and Bob both have access to the key, but Eve doesn't. The black boxes input plaintext and key and output ciphertext (in Alice's case), or input ciphertext and key and output plaintext (in Bob's).

The terms in the figure have the following meanings.

Plaintext: The plaintext is not quite the same as the message being sent. The message probably has to be translated into some standard form to be encrypted; for example, this might be leaving out the punctuation, turning it into ASCII code or a sequence of numbers, etc. But there is nothing secret about this stage; knowing the plaintext is equivalent to knowing the message.

Ciphertext: The ciphertext is what is actually transmitted. In general Alice and Bob must assume that Eve can get her hands on the ciphertext, and they must design the system so that this will not enable her to recover the plaintext.

Key: The encryption uses some extra information, known as the key, which can be varied from one transmission to another. Both Alice and Bob must have information about the key, in order to perform the encryption and decryption.

There are three main types of encryption method:

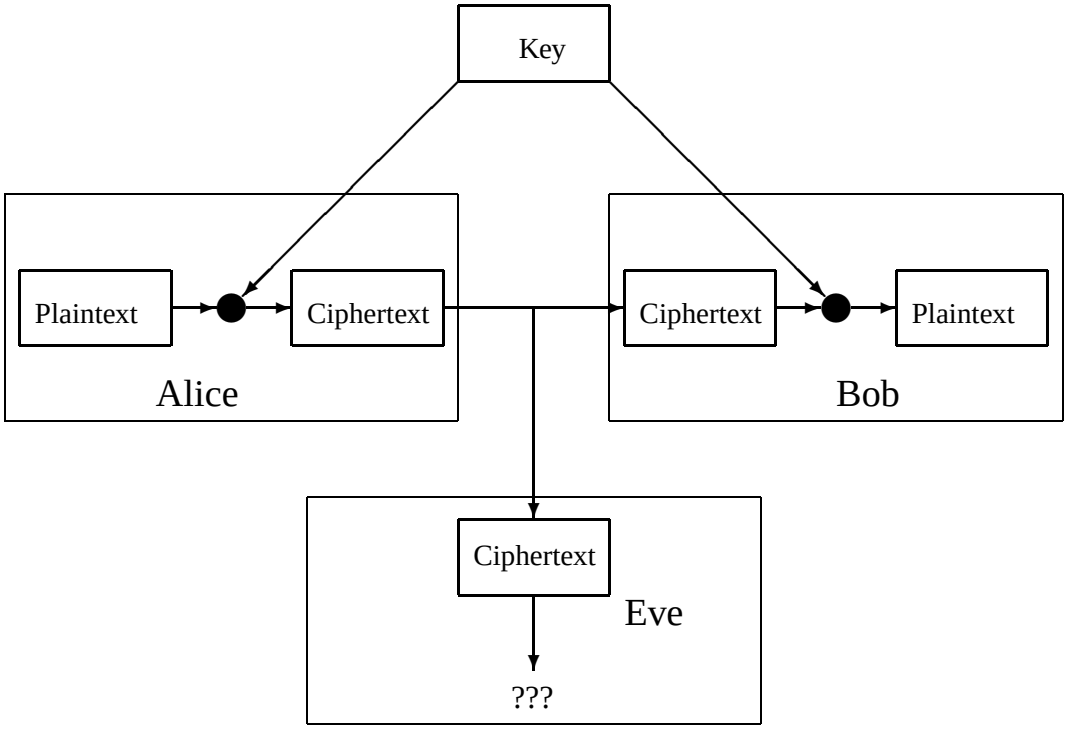


Figure 1.1: The set-up

Transposition: The order of the letters in the plaintext is rearranged in some systematic way. The key is the permutation applied to the positions.

Substitution: Individual letters are replaced by different letters in a systematic way. This may be more complicated than just a single permutation; we may apply different permutations to the letters in different positions. The key is the sequence of applied permutations.

Codebook: Complete words in the message are replaced by other words with quite different meanings. The key is the codebook, the list of words and their replacements.

Of course, the types are not completely separate, and some or all of them can be used together.

Note on the word "code" This word is used with many different meanings in communication theory. Often it just means a scheme for translating information

from one format to another. Thus, for example, the Morse code (used in early telegraph and radio communication) would translate the word "Code" into the sequence

$$-\cdot-\cdot \quad --- \quad -\cdot\cdot \quad \cdot$$

of dots and dashes, while seven-bit ASCII (used in computer communication and representation of data) would translate it into the four numbers 67, 111, 100, 101, or, in binary notation,

$$1000011110111111001001100101$$

An error-correcting code translates a string of symbols into a different string for the purposes of error correction. For example, a $[7,4]$ code might translate 1010 into 1010101.

The term "secret code" might mean what we have called a cipher system, or perhaps a cryptogram (the result of encrypting a message using a cipher system).

Within cryptography, a code replaces certain key words in the message by other words or combinations of symbols, as specified in the code book. This is sometimes contrasted with a cipher, which operates on the individual letters or symbols.

Pig-Latin

Pig-Latin is a simple form of transposition cipher with a "null" character. These rules are taken from the Pig-Latin homepage at

`http://www.idioma-software.com/pig/home.htm`.

For words which begin with a single consonant take the consonant off the front of the word and add it to the end of the word. Then add ay after the consonant. Here are some examples:

$$\begin{aligned} \text{cat} &= \text{atcay} \\ \text{dog} &= \text{ogday} \\ \text{simply} &= \text{implysay} \\ \text{noise} &= \text{oisnay} \end{aligned}$$

For words which began with double or multiple consonants take the group of consonants off the front of the word and add them to the end, adding ay at the very

end of the word. Here are some examples:

$$\begin{aligned} \text{scratch} &= \text{atchscray} \\ \text{thick} &= \text{ickthay} \\ \text{flight} &= \text{ightflay} \\ \text{grime} &= \text{imegray} \end{aligned}$$

For words that begin with a vowel, just add yay at the end. For example:

$$\begin{aligned} \text{is} &= \text{isyay} \\ \text{apple} &= \text{appleyay} \\ \text{under} &= \text{underyay} \\ \text{octopus} &= \text{octopusyay} \end{aligned}$$

A sample of pig-Latin:

> Igpay-Atinlay opensyay upyay ayay ewnay orldway atthay ouyay evernay ouldway avehay oughtthay ossiblepay. Ybay usingyay Igpay-Atinlay, ouyay ootay ancay ulfillfay ouryay ascinatingfay uturefay unctionsfay otay ethay ullestfay ullnessfay astfay. Ouyay illway ebay ayay etterbay ersonpay, avehay ayay etterbay exsay ifelay, andyay ebay etterbay anthay ouryay eighborsnay.

Exercises

1.1. (a) Explain in your own words the meaning of the terms *cryptography*, *cryptanalysis*, and *steganography*.

(b) You want to send a postcard to your family, which will contain a secret message to your brother. How might you do it?

Chapter 2

Substitution ciphers

In the simplest (monoalphabetic) type of substitution cipher, we take a permutation of the alphabet in which the plaintext is written, and substitute each symbol by its image under the permutation. The key to the cipher is the permutation used; anyone possessing this can easily apply the inverse permutation to recover the plaintext.

If we take a piece or ordinary English text, ignore spaces and punctuation, and convert all letters to capitals, then the alphabet consists of 26 symbols, and so the number of keys is

$$26! = 403291461126605635584000000.$$

This is a sufficiently large number to discourage anyone making an exhaustive test of all possible keys. However, the cipher is usually very easy to break, as we will see.

We can represent a permutation by writing down the letters of the alphabet in the usual order, and writing underneath each letter its image under the permutation. To find the inverse, write the bottom row above the top row, and then sort the columns so that the new top row is in its natural order. For example, the inverse of the permutation

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

is

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

The identity permutation is the very simple permutation which leaves each symbol where it is: not much use for enciphering!

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

Finally, the composition $g \circ h$ of two permutations is obtained by applying first g and then h to the alphabet.

Definition A set G of permutations forms a *group* if

(a) for all $g, h \in G$, $g \circ h \in G$;

(b) the identity permutation e belongs to G;

(c) for every $g \in G$, the inverse permutation g' belongs to G.

The *order* of the group G is the number of permutations it contains.

For example, the set of all permutations of an n-element set is a group, called the *symmetric group* of degree n and denoted by S_n. Its order is $n!$. The symmetric group S_n is the set of keys for substitution ciphers with an n-letter alphabet.

2.1 Caesar cipher

The simplest possible substitution cipher is the *Caesar cipher*, reportedly used by Julius Caesar during the Gallic Wars. Each letter is shifted a fixed number of places to the right. (Caesar normally used a shift of three places). We regard the alphabet as a cycle, so that the letter following Z is A. Thus, for example, the table below shows a right shift of 5 places.

```
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
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
```

The message "Send a hundred slaves as tribute to Rome" would be enciphered as `Xjsi f mzsiwji xqfajx fx ywngzyj yt Wtrj`. The key is simply the number of places that the letters are shifted, and the cipher is decrypted by applying the shift in the opposite direction (five places back).

Some practical details make the cipher harder to read. In particular, it would be sensible to ignore the distinction between capital and lower case letters, and also to ignore the spaces between words, breaking the text up into blocks of standard size, for example

```
XJSIF MZSIW JIXQF AJXFX YWNGZ YJYTW TRJXX
```

(We have filled up the last block with padding.)

The Caesar cipher is not difficult to break. There are only 26 possible keys, and we can try them all. In this case we would have

```
XJSIF MZSIW JIXQF AJXFX YWNGZ YJYTW TRJXX
YKTJG NATJX KJYRG BKYGY ZXOHA ZKZUX USKYY
ZLUKH OBUKY LKZSH CLZHZ AYPIB ALAVY VTLZZ
...
SENDA HUNDR EDSLA VESAS TRIBU TETOR OMESS
...
```

Almost certainly only one of the twenty-six lines will make sense, and it is easy to break it into words and discard the padding.

There are other tricks that can be used, which will be important later. As we will see in the next section, in English text, the commonest letter is usually E. Also, the consecutive letters R, S, T, U are common, and are followed by a block V, W, X, Y, Z of relatively uncommon letters. If we can spot these patterns, then we can make a guess at the correct shift. Our example is too short to show much statistical regularity; but (if we assume that the last two Xs are padding) the commonest letter is J, and the letters W, X, Y, Z are common while A, B, C, D, E are rare, so we would guess that the shift is 5 (which happens to be correct). We will look at this again in the next section.

We will in future use the convention that the plaintext is in lower case and the ciphertext in capitals.

A famous modern instance of a Caesar shift was HAL, the rogue computer in the science-fiction story *2001: A Space Odyssey*. The computer's name is a shift of IBM. (The author, Arthur C. Clarke, denied that he had deliberately done this.)

The Caesar shifts form a group. If the alphabet is $A = \{a_0, a_1, \ldots, a_{q-1}\}$, then the shift by i places can be written as $f_i : a_j \mapsto a_{j+i \bmod q}$, and we have

$$\begin{aligned} f_{i_1} \circ f_{i_2} &= f_{i_1+i_2 \bmod q}, \\ f_0 &= e, \\ f_i' &= f_{-i \bmod q}. \end{aligned}$$

The order of this group is q.

2.2 Letter frequencies

In any human language (and in most artificial languages as well), words are not random combinations of symbols, and so they will show various statistical regularities. For example, in English, the commonest letter is E; in a typical (not too short) piece of English, about 12% of all the letters will be E.

As an example, in the text of *Alice's Adventures in Wonderland*, by Lewis Carroll (AAIW for short), the frequencies of the letters (ignoring spaces and punctuation) are given in Table 2.1 (the figure given is the average number of occurrences among 100 letters), in the column labelled "AAIW". (The figures in the table are the average numbers of occurrences among 100 letters of text.) The columns labelled "Meaker" and "Garrett" are from the books *Cryptanalysis* by Helen Fouché Gaines, and *Making, Breaking Codes* by Paul Garrett. Gaines (whose book was published in 1939) took the numbers from a table by O. P. Meaker; Garrett, on the other hand, simply analysed a megabyte of old email. The French and Spanish statistics are also quoted by Gaines, from tables by M. E. Ohaver, *Cryptogram Solving*. The last column will be explained later.

Note that even for English text the figures vary, though not too much: in AAIW the most frequent letters, in order, are E, T, A, O, I, H, N, S, R, D, L, U; in Gaines' table, the order is E, T, A, O, N, I, S, R, H, L, D, U. However, in other languages the order is quite different. For example, in German, the order is typically E, N, I, R, S, A, D, T, U, G, H, O.

Figure 2.1 shows a histogram of the expected frequencies, together with the actual letter frequencies in the message encrypted by Caesar's cipher. It is clear by eye that the best fit is obtained if the actual message is shifted five places left.

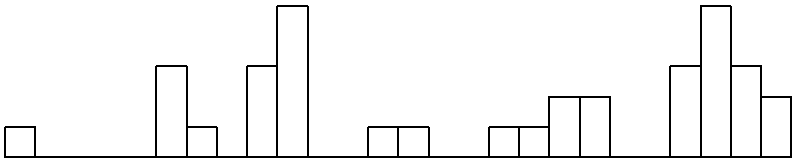
Pairs of letters (referred to as *digrams*) also have their characteristic frequencies. Some of the most common in English are given in Table 2.2. Meaker's tables, and those of Pratt and Fraprie, are taken from Gaines.

One can also analyse trigrams, or longer sequences. Among the most common trigrams in English are THE, ING, THA, AND, ION.

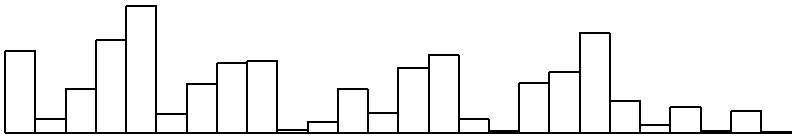
As an indication of how these frequencies reflect the language, here are three "random" pieces of text. In each case, in order to split the text into words, a 27-letter alphabet (consisting of the 26 letters and the space character) has been used; any punctuation characters in the original text are regarded as spaces, and a string of spaces is reduced to a single space. In the first piece of text, the computer has generated random text using the same letter and space frequencies as in AAIW. In the second, the digram frequencies have been used; and in the third, trigram

Letter	AAIW	Meaker	Garrett	French	Spanish	Gadsby
A	8.15	8.05	7.73	9.42	12.69	10.96
B	1.37	1.62	1.58	1.02	1.41	2.14
C	2.21	3.20	3.06	2.64	3.93	2.66
D	4.58	3.65	3.24	3.38	5.58	4.12
E	12.61	12.31	11.67	15.87	13.15	0.00
F	1.86	2.28	2.14	0.95	0.46	2.15
G	2.36	1.61	2.00	1.04	1.12	3.61
H	6.85	5.14	4.52	0.77	1.24	4.91
I	6.97	7.18	7.81	8.41	6.25	8.81
J	0.14	0.10	0.23	0.89	0.56	0.23
K	1.07	0.52	0.79	0.00	0.00	1.18
L	4.37	4.03	4.30	5.34	5.94	5.32
M	1.96	2.25	2.80	3.24	2.65	2.07
N	6.52	7.19	6.71	7.15	6.95	8.61
O	7.58	7.94	8.22	5.14	9.49	10.42
P	1.40	2.29	2.34	2.86	2.43	1.91
Q	0.19	0.20	0.12	1.06	1.16	0.05
R	5.02	6.03	5.97	6.46	6.25	4.77
S	6.05	6.59	6.55	7.90	7.60	6.97
T	9.93	9.59	9.53	7.26	3.91	8.50
U	3.22	3.10	3.21	6.24	4.63	4.16
V	0.78	0.93	1.03	2.15	1.07	0.31
W	2.49	2.03	1.69	0.00	0.00	2.80
X	0.13	0.20	0.30	0.30	0.13	0.04
Y	2.11	1.88	2.22	0.24	1.06	3.18
Z	0.07	0.09	0.09	0.32	0.35	0.11

Table 2.1: Letter frequencies



Actual



Expected

Figure 2.1: Expected and actual letter frequencies in Caesar cipher

frequencies are used. Notice how the random texts resemble the original more closely as longer sequences are used.

Letter frequencies

garyrndtdbleayir hedryeeabeslt tyt watat vnot sooannaheoynoc hhh ndn e n mom scie cehealiiea yneuries u imn h utootpn eomvtet ia ecadehatyba eub e lsrv utl ecnrhmer etwtata nstp thttwttl ht tth dg uyatnpbs toinhpitehttestttthotrehushilwlhtaehyto rovt aget eaeaflrwu gnat asrl eeri luikghreborelephre hhvde egnso nodieiha dcoeothgoa tsabns s cneo ndnhfbtsont ne cpnoed m t old fzl rohuiinirtosthe arrn genialendtr hhntn tsmtr osnol ngohne aiauumnie p hhb te t gtt o araswc tak omlhidtaoi er rlumh ceca tlo acnimal tto sosi ah htoe c sty laaahsouseshi oae oh afasth wnsihnaeoawoi aesnhi yb vresptn gas elplteot or annner en s e dfhat tso nmlr te euhdre ltsnsr f reesd s cchtehavns uhtiwalo tahot lrrnnt

Digraph	AAIW	Meaker	P & F	Garrett
TH	3.23	3.51	3.16	3.18
HE	3.23	2.51	1.08	2.17
AN	1.48	1.72	1.08	1.59
IN	1.89	1.69	1.57	2.59
ER	1.68	1.54	1.33	1.95
RE	1.07	1.48	1.25	1.85

Table 2.2: Frequencies of common digrams

Digram frequencies

tre wherrltau ar a inor hee ly goove aye abinglothased as an nontte fin whike it im yon coveng a per weker ligo d ated ay s red ase ous andldrthi i anory acke owhalist the w an thi tuth abinwaly lyton bofforyilenour t n ns art asod h athostugir telidademifure bing hee hedertliryouricell araks edshe capl asove a asino thaf ar at heldryirry id and aghanorsith anesance age angh oum st athed w waronoubit ir bellea a d a at alle t quceendld hello ag t we mar ncerin avesabout ag thedoed sherkishe ano ai t ithe alkeyorated abomor p rs he ag a itainokittina acerr s abupped iranchendl whecthede awhe athai asus oo i and a s shermfu bar and a thre mer s aig it at a an y b alerd a taryouga shed f aithon iseal anghetheme as put m s n d

Trigram frequencies

ithe pits as but she i hat she peasessid to this begit a said to yout ands i loome four shone shemalice cou at sion to one se al the sped ithe gand nerse shereaverybottly embecon unnoth there pen the droqueelf land gloorger an tol the came in go the could ner so des on a wit ite bee ot the spearep onfor hown aft she is ander han ithe quive cut of ano mut andly wit it wrilice dookinam ther heseen everse ter and owles a saing alice way le jusishe s to its torrock ing teopersed show as dif to happen theirs itte heam whis way vered ant his a sairs handeauteree way murse begs a as sid s yout of ence wo cho and th ord des ned be that speopead the timessizaris ank th all guittelf to holl his and execin hand th t

2.3 Breaking a substitution cipher

Breaking a cipher is an art; it cannot be done by applying a formula. But there are some rules to follow when doing this job. Here is a partly worked example of breaking a substitution cipher; you should complete the working.

The ciphertext is:

```
RZOLB QJOWW QBWIR DQFQE VICOB OKOLR UVIDW QFMRO IVTOH
OVZMA UFUIR UVEWM DWOBH UOVYO RQRZO UBWRM TOVRW RZOSZ
ITRQW COIBQ DOTUO VYORQ RZOWR MTOVR BOYRQ BWIVT RQRZO
WRMTO VRAIT OWRIR MROWC ZUYZD QBOHO BSZIB TFSML QVRZO
ARZOL BQJOW WQBCI WJUVO TUJZO DOEIV ZUWRO IYZUV EIAUV
MROFI ROQBY QVRUV MOTIA UVMRO FQVEO BRZIV RZOJU XOTRU
AOIVT WZQMF TRZUW ZILLO VRZOW RMTOV RWCZQ JIUFO TRQFO
IHORZ OFOYR MBOBQ QAUAA OTUIR OFSCO BORZO AWOFH OWJUV
OTUVI TTURU QVRZO LBQJO WWQBC IWJUV OTUJZ OWZUB KOTOX
LFIUV UVEIT UJJUY MFRLI WWIEO QBUJZ OJIUF OTRQE ORRZB
QMEZR ZOWSF FIDMW ZOCIW JUVOT UJZOF OJRRZ OYURS JQBIT
ISCUR ZQMRR UOBOY RQBWL OBAUW WUQVI VTUJZ OAIBB UOTCI
WIFFQ COTQV FSQVO TISQJ JJQBR ZOLMB LQWOR ZOYUR SJQBU
RWLIB RRQQK IZIVT UVYQV RBQFF UVERZ OLBQJ OWWQB WIVTR
ZOSCO BOJQB YOTRQ RIKOI VQIRZ VQRRQ FOIHO DQFQE VIUVW
OIBYZ QJAQB OFMYB IRUHO QBFOW WQVOB QMWLQ WRWXX
```

We first count the frequencies of the letters. The commonest of the 715 letters, with their frequencies, are given in the table.

O	R	Q	I	U	W	V	B	Z
99	72	59	50	49	48	45	43	43

We also notice that RZ is a very common digram, with 23 occurrences. So we might guess the following identifications: `O = e, R = t, Z = h`. This gives

```
theLB QJeWW QBWIt DQFQE VICeB eKeLt UVIDW QFMte IVTeH
eVhMA UFUIt UVEWM DWeBH UeVYe tQthe UBWtM TeVtW theSh
ITtQW CeIBQ DeTUe VYetQ theWt MTeVt BeYtQ BWIVT tQthe
WtMTe VtAIT eWtIt MteWC hUYhD QBeHe BShIB TFSML QVthe
AtheL BQJeW WQBCI WJUVe TUJhe DeEIV hUWte IYhUV EIAUV
MteFI teQBY QVtUV MeTIA UVMte FQVEe BthIV theJU XeTtU
AeIVT WhQMF TthUW hILLe VtheW tMTeV tWChQ JIUFe TtQFe
```

```
IHeth eFeYt MBeBQ QAUAA eTUIt eFSCe Bethe AWeFH eWJUV
eTUVI TTUtU QVthe LBQJe WWQBC IWJUV eTUJh eWhUB KeTeX
LFIUV UVEIT UJJUY MFtLI WWIEe QBUJh eJIUF eTtQE etthB
QMEht heWSF FIDMW heCIW JUVeT UJheF eJtth eYUtS JQBIT
ISCUt hQMtt UeBeY tQBWL eBAUW WUQVI VTUJh eAIBB UeTCI
WIFFQ CeTQV FSQVe TISQJ JJQBt heLMB LQWet heYUt SJQBU
tWLIB ttQQK IhIVT UVYQV tBQFF UVEth eLBQJ eWWQB WIVTt
heSCe BeJQB YeTtQ tIKeI VQIth VQttQ FeIHe DQFQE VIUVW
eIBYh QJAQB eFMYB ItUHe QBFeW WQVeB QMWLQ WtWXX
```

The other common letters probably include `a`, `i`, `o` and `n`. Various clues help us to make the right identification. For example, consider the string `tQthe`, which occurs several times. Here, `the` is probably either a word or the beginning of a word like `then`. If this is right, `tQ` ends a word, and the most likely possibility is that `Q = o`.

Another clue is that `WW` occurs four times in the text. Double letters are not very common in English; `ee`, `ll` and `ss` are the most common, so probably `W = s`.

After a certain amount of guesswork of this sort, we begin to recognise more complicated words, and we find eventually that the substitution is

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

and the message is

> The professors at Bologna were kept in absolute and even humiliating subservience to their students. They had to swear obedience to the student rectors and to the student-made statutes, which bore very hardly upon them. The professor was fined if he began his teaching a minute late or continued a minute longer than the fixed time, and should this happen the students who failed to leave the lecture-room immediately were themselves fined. In addition, the professor was fined if he shirked explaining a difficult passage, or if he failed to get through the syllabus; he was fined if he left the city for a day without the rector's permission, and if he married, was allowed only one day off for the purpose. The city, for its part, took a hand in controlling the professors, and they were forced to take an oath not to leave Bologna in search of more lucrative or less onerous posts.

This description of employment conditions for academics in the Middle Ages is taken from J. D. Knowles, *The Evolution of Mediaeval Thought*.

Two fictional accounts of substitution ciphers are the stories “The Gold Bug”, by Edgar Allen Poe, and “The Adventure of the Dancing Men”, a Sherlock Holmes story by Sir Arthur Conan Doyle.

Worked example Solve the following substitution cipher.

```
)}&@^ {;‘?@ (‘@,( ^{?}# $‘{+^ ‘;#:^ ,(‘@? }#‘:^
;[^‘= ){*‘! }#@‘{ %^.[: ^;;)@ ){{#+ !^:;? }#={}
,;}+^ @(){* ‘!}#@ )@!#@ ,(^{? }#$‘{ {}@+^ ‘;#:^
)@,(^ {?}#$ ‘{{}@ ^.[:^ ;;)@) {{#+! ^:;?} #:={}
,_^%* ^);}& ‘+^‘* :^‘{% #{;‘@ );&‘$ @}:?= ){%..
```

Solution: This cipher is surprisingly difficult, as you will find if you try it for yourself! A hint makes it much easier. The conclusion of the message, `..`, is padding; you are told that the letter used for padding is `x`. This gives a lot of information, since `.` occurs twice in the rest of the message, and `x` is usually preceded by `e` in English; so we guess that `^` is `e`. Now we have the sequence `ex[:e;;` which is probably going to be `express`, giving us three more letters. Now finish the rest yourself!

The moral of this is that a seemingly innocent trait of the cryptographer, such as always using `x` as a filler, may give away crucial information.

2.4 Affine substitutions

The sharp-eyed will have noticed something special about the substitution used here. It maps `a` to `I`, `b` to `D`, `c` to `Y`, and so on; advancing the plain letter one place moves the cipher letter back five places (or forward 21 places). In otherwords, if the letters of the alphabet are numbered from 0 to 25, so that `a` is represented by 0, `b` by 1, ..., `z` by 25, then the substitution takes the form

$$x \mapsto 8 + 21x \pmod{26}.$$

Such a substitution, or the cipher it generates, is called *affine*.

The Caesar shift is a special case of an affine cipher, having the form

$$x \mapsto x + b \pmod{26}$$

for some fixed b. The general form of an affine cipher is

$$x \mapsto ax + b \pmod{26}$$

for some fixed a and b. The advantage is that the key is simple; instead of needing a general permutation of the letters, we only need the numbers a and b mod 26.

What affine ciphers are possible, and how can they be inverted?

First we must decide when an affine substitution is a permutation. Consider the substitution $\theta : x \mapsto ax+b \pmod{n}$. It will fail to be a permutation if there exist two distinct x_1, x_2 with

$$ax_1 + b \equiv ax_2 + b \pmod{n},$$

that is, $ay \equiv 0 \pmod{n}$, where $y = x_2 - x_1$. So θ is a permutation if and only if the congruence $ay \equiv 0 \pmod{n}$ has a solution $y \not\equiv 0 \pmod{n}$.

Let d be the greatest common divisor of a and n. Then, say, $a = a_1 d$ and $n = n_1 d$ for integers a_1, n_1. Suppose that $d > 1$, so that $n_1 < n$. Putting $y = n_1$, we have

$$ay = a_1 d n_1 = a_1 n \equiv 0 \pmod{n},$$

so θ fails to be a permutation.

Conversely, suppose that $d = \gcd(a,n) = 1$. By Euclid's Algorithm (see the end of this chapter), there exist integers u, v such that $au + nv = 1$. Now, if $ay \equiv 0 \pmod{n}$, then

$$y = (au+nv)y = u(ay) + n(vy) \equiv 0 \pmod{n},$$

so θ is a permutation.

We conclude:

Theorem 2.1 *The affine map $x \mapsto ax+b$ is a permutation if and only if* $\gcd(a,n) = 1$.

What happens if we compose two such maps? Write $\theta_{a,b}$ for the map $x \mapsto ax+b \pmod{n}$, where $\gcd(a,n) = 1$. We have

$$\theta_{a,b} \circ \theta_{a',b'} : x \mapsto ax+b \mapsto a'(ax+b)+b',$$

so $\theta_{a,b} \circ \theta_{a',b'} = \theta_{aa',ba'+b'}$.

The identity permutation $x \mapsto x$ is the map $\theta_{1,0}$. So to find the inverse of $\theta_{a,b}$ in the form $\theta_{a',b'}$, we have to solve the congruences

$$\begin{aligned} aa' &\equiv 1 \pmod{n}, \\ ba' + b' &\equiv 0 \pmod{n}. \end{aligned}$$

The first congruence has a unique solution mod n, which can be found by Euclid's Algorithm as before. Then the second congruence also has a unique solution, namely $b' \equiv -ba' \pmod{n}$.

In particular, with $n = 26$, we want to invert the map $\theta_{21,8}$. By trial and error (or by Euclid's Algorithm), $21 \cdot 5 \equiv 1 \pmod{26}$; and then $-5 \cdot 8 \equiv 12 \pmod{26}$. So the inverse of $\theta_{21,8}$ is $\theta_{5,12}$.

Definition *Euler's totient function* ϕ is the function on the natural numbers given by

$$\phi(n) = \begin{cases} \text{number of congruence classes } a \bmod n \\ \text{such that } \gcd(a,n) = 1. \end{cases}$$

We give a formula for it, which will be proved later.

Theorem 2.2 *Let $n = p_1^{a_1} p_2^{a_2} \cdots p_r^{a_r}$, where $p_1, p_2, \ldots, p_r$ are distinct primes and $a_1, a_2, \ldots, a_r > 0$. Then*

$$\phi(n) = p_1^{a_1-1}(p_1-1)p_2^{a_2-1}(p_2-1)\cdots p_r^{a_r-1}(p_r-1).$$

For example, $26 = 2 \cdot 13$, so $\phi(26) = 1 \cdot 12 = 12$. The congruence classes coprime to 26 are represented by the odd numbers from 1 to 25 excluding 13.

Theorem 2.3 *The set of affine permutations mod n is a group of order $n \cdot \phi(n)$.*

We have verified the group properties in the earlier argument. For the order, note that there are $\phi(n)$ choices for a and n choices for b.

There are thus $26 \cdot 12 = 318$ affine permutations. If we know or suspect that a substitution cipher is affine, we could try all 318 keys, though this is not trivial by hand. The method of looking for patterns of consecutive letters does not apply. Like any substitution cipher, an affine cipher is vulnerable to frequency analysis. Its advantage is the small size of the key (two numbers rather than a complete permutation.)

Worked example Decrypt the following affine substitution cipher:

```
JZQOU DQGKZ UULYU MKUOX LQJQJ ZQZCW ZQDYU MDXUJ
QRJCE LQEDR CRWGL UUIEJ JZQEP QDEWQ QEDRC RWGCR
JZCGK ZEDJJ ZQYJQ LLJZQ GJUDY
```

You are given that the frequency distribution in the ciphertext is as follows:

C	D	E	G	I	J	K	L	M	O	P	Q	R	U	W	X	Y	Z
6	8	7	5	1	13	3	6	2	2	1	15	6	10	4	2	4	10

Solution The commonest letter Q in the given cipher is likely to be e. We also see that the trigram JZQ occurs five times and so is likely to be the. This gives J=t and Z=h.

The letters Q and Z are x_{16} and x_{25} (where $q = 26$ here), while e and h and x_4 and x_7. Thus the parameters c and d satisfy

$$\begin{aligned} 4c+d &\equiv 16 \pmod{26}, \\ 7c+d &\equiv 25 \pmod{26}, \end{aligned}$$

from which we find $c = 3$ and $d = 4$. From this the entire substitution can be worked out, and we find the plaintext to be

```
themo resch oolyo ucomp letet hehig heryo urpot
entia learn ingsl ookat theav erage earni ngsin
thisc hartt heyte llthe story
```

or, correctly spaced and with punctuation,

> The more school you complete, the higher your potential earnings. Look at the average earnings in this chart; they tell the story!

2.5 Making a substitution cipher safer

A substitution cipher can be solved by frequency analysis, and so is insecure for all but the shortest messages. However, there are some improvements that can be made. The first two rely on using a different alphabet for the ciphertext, with more characters than the plaintext alphabet. For example we could use an alphabet of 100 characters, represented by symbols $00, 01, \ldots, 99$.

Nulls: These are additional symbols in the cipher alphabet which do not have any meaning but are inserted in random positions to confuse the frequency analysis.

Homophones: We can translate the same letter in plaintext by several different letters in ciphertext. For example, if we use a 100-character cipher alphabet, we can associate about as many characters with each plaintext letter as its percentage frequency in normal text (say, 12 characters for `e`, 9 for `t`, and so on). Then we randomly decide which character to substitute for each occurrence of a letter. In the ciphertext, each character will occur approximately the same number of times. However, the ciphertext is still not random, and patterns of digraphs and trigraphs can be recognised.

Use of language: We can further confuse the analysis by using words from other languages, or by careful choice of words. As an example of what can be done, at least two English novels have been written containing no occurrence of the letter `e`, the commonest letter in English. One of these is *Gadsby*, by Ernest Vincent Wright. The author tied down the `E` key of his typewriter to write the book. The first paragraph reads as follows:

> If youth, throughout all history, had had a champion to stand up for it; to show a doubting world that a child can think; and, possibly, do it practically; you wouldn't constantly run across folks today who claim that "a child don't know anything." A child's brain starts functioning at birth; and has, amongst its many infant convolutions, thousands of dormant atoms, into which God has put a mystic possibility for noticing an adult's act, and figuring out its purport.

To a casual glance, there is nothing odd about this; but it would pose an obvious problem for a cryptanalyst if encrypted with a substitution cipher. A frequency analysis of *Gadsby* is included in Table 2.1.

The novel *A Void* is even more remarkable, having been translated by Gilbert Adair from the French novel *La Disparition* by Georges Perec, which also lacked the letter `e`.

Another trick is to write words "phonetically", or to use text-messaging abbreviations.

Features of text messaging language such as phonetic spelling (such as “nite” for “night”), the common omission of vowels (“txt” for “text”), use of abbreviations (such as AFAIK for “as far as I know”), use of numerals `2`, `4` and `8` for `to`, `for` and `ate`, and use of “emoticons” such as `;-)` as an essential part of the text, would give frequency analysis quite different from standard English. I don’t know whether such analysis of a body of text messages has been done.

Transposition: The substitution can be combined with *transposition*, that is, permuting the order of the characters in the ciphertext in a specified way. This will help to destroy the patterns of digram and trigram frequencies.

With these improvements, even a substitution cipher can be effective for a short message which will not receive very sophisticated analysis.

2.6 Related ciphers

A number of ingenious variants on substitution ciphers have been proposed. Many of these are discussed by Helen Fouché Gaines in the book *Cryptanalysis*. I will describe just one here: the *Playfair cipher*.

The key to this cipher is a single word. Draw a 5×5 grid, and starting in the top left, write the letters of the keyword: a letter occurring more than once is only written on its first occurrence. Then fill the grid with the remaining letters of the alphabet. Since there are 26 letters and only 25 spaces, we regard `I` and `J` as identical for this purpose.

For example, suppose that the keyword is `THOUGHTFULLY`. Then the filled grid is:

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

Now the letters of the message are encrypted two at a time, according to the following rules:

- First, two identical letters are separated by a dummy letter.

- Two letters in the same row of the square are replaced by the letters immediately to their right (with the convention that rows "wrap around", so that to the right of `K` in our square comes `C`, for example).
- Two letters in the same column of the square are replaced by the letters immediately below them (with a similar wrap-around convention).
- Two letters not in the same row or column form two corners of a rectangle; they are replaced by the letters in the opposite corners of the rectangle, where the letter in the same row as the first letter of the plaintext comes first.

Suppose, for example, that we want to encrypt the message "I must see you. Come to the Half Moon at nine", with the keyword `THOUGHTFULLY` as above. Writing the plaintext in pairs, using `x` as a dummy, we get

```
im us ts ex ey ou co me to th eh al fm ox on at
ni ne
```

which is encrypted as

```
CQ TX FT IW PE UG ET PC HU HO DO BY CS UW HP FU
QD PD
```

The message can then be broken up differently to help conceal its origin.

Decryption is done in the same way as encryption, but replacing "right" and "below" by "left" and "above" in the second and third rules. Then two ambiguities must be resolved: first, the choice must be made between `i` and `j`; then, dummy letters must be recognised and removed.

Despite appearances, the Playfair cipher can be regarded as a simple substitution cipher, over the 676-letter alphabet consisting of all digrams; so a sufficiently long message can be broken by statistical techniques. However, it has much more structure resulting from the grid. For example, although any single letter may be replaced by any other, it is most frequently replaced by the letters immediately to its right and below it. Moreover, the letter to the right of a given one has a high probability of being the next letter in the alphabet. Also, if `ab` is encrypted as `CD`, then `ba` is encrypted as `DC`.

For example, if we knew that `HU HO` encrypts `to th`, we could infer that some row or column of the grid contains the consecutive letters `THOU`. We sould guess that this combination occurs in the keyword (probably at the start).

A worked example of breaking a Playfair cipher using a short crib is given in the detective story *Have His Carcase*, by Dorothy L. Sayers.

2.7 Number theory

In this section we give more details of some of the number theory which underlies our discussion of affine ciphers.

Euclid's algorithm

Euclid's algorithm is a procedure to find the greatest common divisor of two integers. In the form of a one-line recursive program it can be written as follows:

`if` $b = 0$ `then` $\gcd(a,b) := a$ `else` $\gcd(a,b) := \gcd(b, a \bmod b)$ `fi`

where $a \bmod b$ means the remainder on dividing a by b.

For example,

$$\gcd(30,8) = \gcd(8,6) = \gcd(6,2) = \gcd(2,0) = 2.$$

The algorithm can also be used to write $\gcd(a,b)$ in the form $ua+vb$ for some integers u,v. We express each successive remainder in this form until we reach the last non-zero remainder, which is the gcd. In the above example,

$$\begin{aligned} 6 &= 30 - 3\cdot 8 \\ 2 &= 8 - 1\cdot 6 \\ &= 8 - (30 - 3\cdot 8) \\ &= (-1)\cdot 30 + 4\cdot 8, \end{aligned}$$

so $u = -1$, $v = 4$.

This can be used to find inverses mod n. For example, $\gcd(21,26) = 1$, and Euclid's algorithm shows that $1 = (-4)\cdot 26 + 5\cdot 21$; so $5\cdot 21 \equiv 1 \pmod{26}$, and the inverse of 21 mod 26 is 5.

Euler's function

In this section we prove Theorem 2.2. We begin with the theorem known as the *Chinese Remainder Theorem*.

The following discussion is based on the section on Chinese mathematics in George Gheverghese Joseph, *The Crest of the Peacock: Non-European Roots of Mathematics*, Penguin Books 1992. The fourth-century text *Sun Tsu Suan Ching* (Master Sun's Arithmetic Manual) contains the following problem:

> There is an unknown number of objects. When counted in threes, the remainder is 2; when counted in fives, the remainder is 3; when counted in sevens, the remainder is 2. How many objects are there?

The problem asks for an integer N such that $N \equiv 2 \pmod 3$, $N \equiv 3 \pmod 5$, and $N \equiv 2 \pmod 7$. One solution is given as

$$N = 2 \cdot 70 + 3 \cdot 21 + 2 \cdot 15 = 233;$$

it is clear that adding or subtracting a multiple of 105 from any solution gives another solution; so the smallest solution is

$$N = 233 - 2 \cdot 105 = 23.$$

A folk-song gives the mnemonic:

> Not in every third person is there one aged three score and ten,
> On five plum trees only twenty-one boughs remain,
> The seven learned men meet every fifteen days,
> We get our answer by subtracting one hundred and five over and over again.

Why does it work? Observe that 70 is congruent to 1 mod 3, to 0 mod 5, and to 0 mod 7, and similarly for 21 and 15; then $70a + 21b + 15c$ is congruent to a mod 3, to b mod 5, and to c mod 7, as required.

A similar procedure works in general. We give the result just for two moduli: it is easily extended to any number by induction.

Let $\mathbb{Z}/(n)$ denote the set of congruence classes mod n. It is clear that, if $x \equiv x' \pmod{mn}$, then $x \equiv x' \pmod m$; so, for $x \in \mathbb{Z}/(mn)$, there is a well-defined element x mod m of $\mathbb{Z}/(m)$. Similarly with n replacing m.

Theorem 2.4 (Chinese Remainder Theorem) *If* $\gcd(m,n) = 1$, *then the map F from $\mathbb{Z}/(mn)$ to $\mathbb{Z}/(m) \times \mathbb{Z}/(n)$ defined by*

$$F(x) = (x \bmod m, x \bmod n)$$

is a bijection.

Proof: Suppose that $F(x) = F(x')$. Then $x \bmod m = x' \bmod m$, that is, m divides $x - x'$. Similarly n divides $x - x'$. Since m and n are coprime, it follows that mn divides $x - x'$, so that $x = x'$ (as elements of $\mathbb{Z}/(mn)$). Thus F is one-to-one.

Now $|\mathbb{Z}/(mn)| = mn = |\mathbb{Z}/(m) \times \mathbb{Z}/(n)|$; so F must also be onto, and thus a bijection.

This proof is non-constructive, whereas the original Chinese argument gave an algorithmic way to compute the inverse of F. This can be generalised as follows. Since $\gcd(m,n) = 1$, Euclid's algorithm shows that there exist numbers a and b such that $am + bn = 1$. Now we see that

$$\begin{aligned} am &\equiv 0 \pmod{m}, \qquad am \equiv 1 \pmod{n}, \\ bn &\equiv 1 \pmod{m}, \qquad bn \equiv 0 \pmod{n}, \end{aligned}$$

so the solution to the simultaneous congruences

$$x \equiv y \pmod{m}, \qquad x \equiv z \pmod{n}$$

is given by

$$x \equiv bny + amz \pmod{mn}.$$

Remark: In fact F is a *ring isomorphism*: this simply means that $F(x+x') = F(x) + F(x')$ and $F(xx') = F(x)F(x')$.

Now $\gcd(x, mn) = 1$ if and only if $\gcd(x,m) = 1$ and $\gcd(x,n) = 1$. In other words, if $F(x) = (y,z)$, then $\gcd(x,mn)$=1 if and only if $\gcd(y,m) = 1$ and $\gcd(z,n) = 1$. Since Euler's function gives the number of congruence classes coprime to the modulus, the Chinese Remainder Theorem implies that

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

if $\gcd(m,n) = 1$.

This extends to products of any number of pairwise coprime factors. Thus

$$\phi(p_1^{a_1} \cdots p_r^{a_r}) = \phi(p_1^{a_1}) \cdots \phi(p_r^{a_r})$$

if $p_1, \ldots, p_r$ are distinct primes.

So, to complete the proof of the theorem, we have to show only that $\phi(p^a) = p^{a-1}(p-1) = p^a - p^{a-1}$ for p prime and $a > 0$. This holds because, of the p^a congruence classes mod p^a, the ones not coprime to p^a are precisely those divisible by p, of which there are p^{a-1}.

Exercises

2.1. Mpuk hu Lunspzo dvyk, woyhzl, vy zlualujl dopjo pz ayhuzmvytlk puav huvaoly Lunspzo dvyk, woyhzl, vy zlualujl dolu zvtl Jhlzhy zopma pz hwwsplk av pa. Aol svunly fvby woyhzl, aol tvyl thyrz fvby huzdly dpss yljlpcl.

2.2.

The following problem is taken from Chin Chiu Shao's book *Su Shu Chiu Chang* (Nine Sections of Mathematics), written in 1247. A ko is a unit of volume.

> Three thieves, A, B and C, entered a rice shop and stole three vessels filled to the brim with rice but whose exact capacity was not known. When the thieves were caught and the vessels recovered, it was found that all that was left in Vessels X, Y and Z were 1 ko, 14 ko and 1 ko respectively. The captured thieves confessed that they did not know the exact quantities they had stolen. But A said that he had used a horse ladle (capacity 19 ko) and taken the rice from X. B confessed to using his wooden shoe (capacity 17 ko) to take the rice from vessel Y. C admitted that he had used a bowl (capacity 12 ko) to help himself from the rice from vessel Z. What was the total amount of rice stolen?

2.3. (a) Solve the simultaneous congruences $x \equiv 4 \pmod{13}$, $x \equiv 5 \pmod{17}$.

(b) Find the inverse of 20 mod 77.

2.4. (a) Show that an affine permutation θ mod q is completely determined if we know its effect on some two different congruence classes mod q.

(b) Show that every permutation in S_q is affine if and only if $q \leq 3$.

(c) For $q = 26$, show that each affine permutation has 0, 2 or 26 fixed points, and that the average number of fixed points is 1. Can you generalise this to any value of q?

Chapter 3

Stream ciphers

Substitution ciphers have been used since time immemorial. As we have seen, they are vulnerable to frequency analysis based on the statistics of the language used. Although frequency analysis was first developed by Arab cryptographers in the tenth century, substitution ciphers continued to be used until quite recently. Simon Singh, in *The Code Book,* tells the dramatic story of how the breaking, by Elizabeth's cryptanalysts, of the cipher used by Mary Queen of Scots led to her trial and execution in 1587. Apparently Mary and her conspirators thought their cipher was secure.

Eventually, it was realised that better ciphers were needed. Many schemes were tried, but the essential idea was to use different substitutions for different letters of the plaintext. The general name of a cipher based on this principle is a *stream cipher*. In this chapter we discuss stream ciphers.

We begin with a general principle, known as *Kerckhoffs' Principle*:

> Alice and Bob must always assume that Eve knows the encryption system they are using, as well as having intercepted the ciphertext. All they can hope to keep secret is the key.

For, although cryptographers continually invent new systems, knowledge of these systems will soon spread in the intelligence community.

3.1 The Vigenère cipher

In 1562, Blaise de Vigenère invented a cipher in which a different Caesar shift is applied to each letter of the plaintext.

Suppose that we shift the first letter by 5, the second by 14, the third by 23, the fourth by 4, and the fifth by 18. Thus the word `enemy` would be encrypted as `JBBQQ`. Notice that the two occurrences of `e` in the original message are replaced by different letters (`J` and `B`). Conversely, different letters in the plaintext become the same in the ciphertext.

The key to this cipher is the sequence $(5, 14, 23, 4, 18)$. Vigenère's idea was that, instead of having to remember the sequence of numbers, it is enough to remember the letters obtained by shifting the letter `a` by these numbers. In this case, `aaaaa` would become `FOXES`; this is the key to the cipher.

We can represent the process by a *Vigenère square*, as shown in Table 3.1. Write down the plaintext with the key immediately under it:

e	n	e	m	y
F	O	X	E	S
J	B	B	Q	Q

Now look in row `e` and column `F` to find the first letter in the ciphertext to be `J`. Repeat for the remaining letters.

What if the message is longer than the key? Vigenère's idea here was to repeat the key as often as necessary:

e	n	e	m	y	p	a	t	r	o	l	s
F	O	X	E	S	F	O	X	E	S	F	O
J	B	B	Q	Q	U	O	Q	V	G	Q	G

So the ciphertext is `JBBQQ UOQVG QG`.

So the key is a simple word or phrase which can be easily memorised and can be changed frequently.

Breaking the Vigenère cipher

The Vigenère cipher is a great advance on the monoalphabetic substitution cipher, and was used for hundreds of years. However, it has two weaknesses, which eventually led to a system of cryptanalysis for it. These are that the cipher applied to each letter is a simple Caesar shift, which is very easy to break, and the fact that the key string repeats after a relatively short number of steps.

Suppose that we knew that the keyword contains five letters. Then we can divide the ciphertext into five strings, where the first string contains the first, sixth, eleventh, ..., letter; the second string contains the second, seventh, twelfth, ...,

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

Table 3.1: Vigenère square

letter; and so on. Now each string is a Caesar cipher and can be attacked by the methods we have already discussed. (We cannot use digram or trigram frequencies here, since letters which are consecutive in one of the substrings were five steps apart in the original message. But the letter frequency analysis, and in particular the frequency patterns of consecutive letters in the alphabet, can be applied.) Once we have a conjectured decryption of each string, we can reassemble them to give the message.

How do we determine the length of the key? We could simply use trial and error. The frequency analysis is not likely to give sensible answers unless the assumed length is a small multiple of the true length.

A more systematic method uses repeats in the ciphertext. A common digram like `th` will probably occur many times in a reasonably long message. If the key length is 5, then the number of different encryptions of it is (at most) 5, and two occurrences will be encrypted in the same way if their positions in the plaintext differ by a multiple of 5. If the key is `FOXES`, then `th` will be encrypted as `YV`, `HE`, `QL`, `XZ`, or `LM`, according as its starting position is congruent to 1, 2, 3, 4 or 5 mod 5.

If we notice that the digram `YV` occurs in positions 1, 66, and 111 of the message, we might guess that it represents `th`, and that the length of the key is a common factor of 65 and 110. Since $\gcd(65, 110) = 5$, we would deduce that the key has length 5. We have more information too: if our guesses are correct, then the first two letters of the key are also revealed as `FO`.

Two digrams could agree by chance, so it is safer to apply the method to trigrams, if we have a reasonable amount of ciphertext.

The first person to propose this method was Charles Babbage, better known as the inventor of the "Difference Engine" and the "Analytical Engine" (two mechanical computers) in the nineteenth century. Babbage never published his decryption method, and Simon Singh speculates that it might have been used by British Intelligence (who would want the method kept secret!) A few years later, Friedrich Kasiski proposed a similar method which now carries his name.

Chi-squared

The method can be mechanised to some extent. We now describe a method for suggesting a solution to a Caesar cipher, which can be applied after we have found the length of the keyword. This uses the *chi-squared statistic*, which statisticians use for measuring the goodness of fit of data. Unlike statisticiaans, we make no

assumptions about the distribution of our data, and draw no conclusions about the significance of the result; the method simply suggests a possible decryption.

It should be stressed that in simple cases, pattern matching by eye is perfectly satisfactory; but it is easier to tell the computer to optimize a complicated function than to do some pattern matching.

Suppose that n objects are put into q boxes, where the probability that each object is put into the ith box is p_i (with $\sum p_i = 1$). The expected number of objects in the ith box is $e_i = np_i$. Suppose that the actual number in the ith box is a_i. Then the chi-squared statistic is

$$X = \sum_{i=1}^{q} \frac{(a_i - e_i)^2}{e_i}.$$

The smaller the value of X, the better the data fit the prediction.

Now suppose we have a piece of text of length n encoded with a Caesar shift, which we want to find. We apply what we hope is the inverse shift to the text. If we are right, then the result should be plaintext, and the letter frequencies should approximate those in English text, that is, $e_i = np_i$, where p_i is the relative proportion of the occurrences of letter i in English. So we calculate the chi-squared statistic, where a_i is the actual number of occurrences of letter i in the shifted text. If we are right, its value will be small. So we try all 26 shifts; the most likely decryption is the one with the smallest value of X.

This method only uses letter frequencies and makes no use of digrams, trigrams, etc. So it can be applied separately to all the substrings of a Vigenère enciphered text, once we know the period.

Here is a worked example. The following is encrypted with a Vigenère cipher with key of length 5.

```
FZFGW BOPFW LWKRA SUQSY JHSIJ DHFVW ICCWA YHFRY GMEIJ
XWPXW WCKXZ JPXRC FBASX MOSMF LBLXZ NBDXG ICLRU JCOXO
NQBWZ JVXHH JSMIV NBQSL MSYSG PVBVK NGQIJ BOPVW FRFRY
GIQML MOARG UWZXM WSPSJ HCKZW WGXXA TBPMF NHXRV BVXXA
XHEIM XSLJS GCLOL MCRKZ YOIMU JKFXZ TIQTA HHRVW XCOGG
SJBVK FHFSF XGLWZ JKXWU TBPMV JFFRY NBEIJ TKKQA SRXWO
JZIEK XVBGG ZZAJG WHEIZ THAEQ ROAIZ JFCIW QJBVQ XZBIH
DOKHK YIMMV BVBXZ JFQLW UZBEK ZFBSX ROHMF LOAEA XMZLS
NBTSM QRYIO TFQLL MSQVG ZPIIG KUBXL NBDYH FBATA HYFRY
YVBHS NGFIK BVBRK ZRAIF QMXAZ NHBVS GPFXO NHETA SYBCW
XFXRU QCPIT DVBV
```

The first of the five substrings that we have to analyse is obtained by taking the first letter of each block; it is
FBLSJDIYGXWJFMLNIJNJJNMPNBFGMUWHWTNBXXGMYJTHXSFXJTJNTS
JXZWTRJQXDYBJUZRLXNQTMZKNFHYNBZQNGNSXQD.
The letter frequencies in this substring are given in the third column of Table 3.2.

We calculate the chi-squared values using the frequency data from *Alice's Adventures in Wonderland*. Table 3.2 gives the calculation for shifts 0 and 5; it is easy to automate this to work out all values.

We find that, for a shift of 5, the value of chi squared is 23.99. The smallest value for any other shift is 281.56, for a shift of 1. This strongly suggests that the shift is 5 and the first letter of the keyword is F.

By the same method (and the results are as clear-cut in all cases), we find the shifts for the other substrings to be $14, 23, 4, 18$, so that the keyword is FOXES. The decrypted text is

> Alice was beginning to get very tired of sitting by her sister on the bank and of having nothing to do: once or twice she had peeped into the book her sister was reading, but it had no pictures or conversations in it, and "what is the use of a book," thought Alice, "without pictures or conversations?" So she was considering, in her own mind (as well as she could, for the hot day made her feel very sleepy and stupid), whether the pleasure of making a daisy-chain would be worth the trouble of getting up and picking the daisies, when suddenly a White Rabbit with pink eyes ran close by her.

In fact, finding the period can also be mechanised to some extent, using a method due to William Friedman. See Garrett's book for a description of this.

3.2 Stream ciphers

The cryptographers now had two tasks. First, they had to find a way of producing a non-repeating key; second, to make the frequency analysis more difficult, they had to use an arbitrary permutation of the alphabet in each position, rather than just a shift. The two tasks require completely different ideas.

These complications also make it much more difficult to use the ciphers, especially in situations such as a battlefield signal unit. Thus it was necessary to move from hand to machine for the encryption and decryption.

Letter	Frequency %	Observed	Expected Shift 0	Expected Shift 5
A	8.15	0	7.58	0.73
B	1.37	5	1.27	2.32
C	2.21	0	2.06	0.12
D	4.58	3	4.26	1.96
E	12.61	0	11.73	0.65
F	1.86	5	1.73	7.58
G	2.36	4	2.20	1.27
H	6.85	3	6.37	2.06
I	6.97	2	6.48	4.26
J	0.14	11	0.13	11.73
K	1.07	1	1.00	1.73
L	4.37	3	4.06	2.20
M	1.96	5	1.82	6.37
N	6.52	11	6.06	6.48
O	7.58	0	7.05	0.13
P	1.40	1	1.30	1.00
Q	0.19	4	0.18	4.06
R	5.02	2	4.67	1.82
S	6.05	4	5.63	6.06
T	9.93	6	9.23	7.05
U	3.22	2	2.99	1.30
V	0.78	0	0.73	0.18
W	2.49	4	2.32	4.67
X	0.13	9	0.12	5.63
Y	2.11	4	1.96	9.23
Z	0.07	4	0.65	2.99
$\sum(o-e)^2/e$			1949.79	23.99

Table 3.2: A chi-squared calculation

One more thing to remember is that we are not restricted to using the Roman alphabet for our ciphers. We can translate our message into a string in any alphabet at all, and use this as the plaintext. In particular, the plaintext could be a string of digits (so that the alphabet is $\{0,1,2,3,4,5,6,7,8,9\}$, or a string of binary digits (so that the alpabet is just $\{0,1\}$.

In the 1930s, a standard International Telegraph Code was agreed (see Figure 3.3). This is based on a code invented by Baudot, whose name has given rise to the word *baud* for the rate of information transmission. The ITC translates the 26 letters and 6 control characters into sequences of length 5 from a two-letter alphabet. With hindsight and familiarity with computers, we regard the symbols of the alphabet as 0 and 1; but originally they were two voltage levels in international telegraphy ($+80$ and -80 volts), or "hole" and "no hole" in punched paper tape. The names of the symbols don't matter, but the names 0 and 1 will be very convenient later.

Using the ITC, a message is encoded into a string of zeros and ones. We can regard this as a string of length $5n$ over the alphabet $\{0,1\}$, or as a string of length n over an alphabet of 32 symbols (the 26 letters and six control characters), whichever is more convenient.

Generating the key

The best key is a completely random sequence of letters from the alphabet. Such a sequence is called a "one-time pad". As we will see later, the one-time pad provides an absolutely secure form of encryption; no possible deductions about the plaintext can be made from knowledge of the ciphertext if this system is used properly.

However, it is very difficult to generate a truly random sequence. (There are rumours that people were employed by the CIA to toss coins all day and write down the results to produce one-time pads for the two-letter alphabet (whose letters might be called "heads" and "tails" in this case). It seems very likely that one-time pads were produced and used by intelligence services. Peter Wright, in *Spycatcher*, records the finding of one-time pads in the personal possessions of suspected Soviet spies in London by MI5 during the Cold War.

The difficulties of producing a random key led to various types of mechanical or electronic devices for producing what are known as "pseudo-random" keys. These are sequences of letters which, although not random, behave in many ways like a random sequence, so that a short sequence of the key gives very little information about the rest of the key. In particular, we require that each letter occurs

A	11000
B	10011
C	01110
D	10010
E	10000
F	10110
G	01011
H	00101
I	01100
J	11010
K	11110
L	01001
M	00111
N	00110
O	00011
P	01101
Q	11101
R	01010
S	10100
T	00001
U	11100
V	01111
W	11001
X	10111
Y	10101
Z	10001
Letters	11111
Figures	11011
Line feed	01000
Carriage return	00010
Word space	00100
All space	00000

Table 3.3: International teleprinter code

with the same frequency, and similarly for digrams, trigrams, etc. We also require that the sequence does not repeat during the transmission of a typical message.

Every deterministic finite machine which outputs a string of characters must eventually repeat; its output will be *ultimately periodic*. That is because the machine must be in one of a finite (possibly very large) number of states at any moment. If it operates continuously, it must eventually return to the same state that it was in at some previous time. From that point on, its behaviour will be the same as on the previous occasion; so the output is periodic. (The period may be very large. For example, a computer with 128 megabytes of memory has 2^{30} transistors, each capable of being in two states; so the number of configurations is $2^{2^{30}}$. In principle, the period could be as large as this number, approximately $10^{300000000}$.)

We will look later at some of pseudo-random number generators which have been used in practice.

Combining key and plaintext

The Vigenère square gives a method of combining plaintext with key to give ciphertext. We can descibe it more simply by identifying the letters `A...Z` with the elements $0 \ldots 25$ of $\mathbb{Z}/(26)$. Then the combination of plaintext letter p and key letter k gives the ciphertext letter $z = p + k$, where the addition is mod 26. Then decrypting simply involves subtraction mod 26: $p = z - k$.

In the Second World War, the Japanese military ciphers often used the digits $0 \cdots 9$ as symbols. The ciphers would also often use a codebook where various commonly used terms were encoded as groups of four digits. Thus, for example, `0700` could refer to the *kōkū tokushi musentai* (Air Special Radio Unit), and `4698` to the *kōkū tokushu jōhōtai* (Air Special Intelligence Unit). The key was a string of pseudo-random digits, and the encryption was addition mod 10, or addition without carrying. Thus, encrypting `4698` with key `7251` would give `1849`. Once again, decryption is subtraction mod 10 (subtraction without borrowing).

The same principle can be used in the simpler case of the binary alphabet. The rules for addition without carry give the addition table of the integers mod 2 (the finite field with two elements, often called the *binary field*:

$+$	0	1
0	0	1
1	1	0

Then, if the plaintext and key are strings of zeros and ones, we just add the mod 2; for example:

Plaintext:	01001001010…
Key:	10100010011…
Ciphertext:	11101011001…

Latin squares

It is possible to generalise the way in which we combine the plaintext and key to form the ciphertext in a stream cipher.

For each character of the key we associate a function mapping plaintext characters to ciphertext characters. This mapping must be a permutation, so that the recipient can invert it to recover the plaintext. So the addition table must have the property that each character appears exactly once in each column.

A Latin square of order q is an $q \times q$ array whose entries are taken from an alphabet of q symbols such that each symbol occurs exactly once in each row and column. This is a stronger requirement than we need; we will see later why it is a good feature from a cryptographic point of view, as we will see later.

In particular, the Vigenère square, the addition table of $0,\ldots,9$ mod 10, and the addition table of the binary field (with the borders removed) are Latin squares. However, there are many other Latin squares. The exact number is not known; it is known that there are upper and lower bounds for the number of Latin squares of order q of the form $(cq)^{q^2}$ for positive constants c.

For example, here is a Latin square of order 10, using the alphabet $\{0,\ldots,9\}$. I have bordered it with row and column indices for ease of use in enciphering.

	0	1	2	3	4	5	6	7	8	9
0	8	6	3	1	2	5	9	7	0	4
1	1	8	4	3	7	0	6	5	9	2
2	4	1	6	2	3	8	0	9	7	5
3	9	3	2	4	0	7	5	1	6	8
4	6	2	5	7	4	1	3	0	8	9
5	0	9	7	6	8	4	1	2	5	3
6	2	7	0	5	6	9	8	3	4	1
7	5	4	9	8	1	2	7	6	3	0
8	7	5	8	0	9	3	2	4	1	6
9	3	0	1	9	5	6	4	8	2	7

(This random Latin square was produced by a Markov chain algorithm due to Jacobson and Matthews.)

Thus, encrypting the plaintext 4698 with key 7251 using this square gives the ciphertext 0065. (For example, the entry in row 4 and column 7 is 0.) A Latin square used in this way is called a *substitution table*. The *columns* of the substitution table are the permutations of the alphabet associated with the key symbols. In the above, the key symbol 0 corresponds to the permutation

$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 8 & 1 & 4 & 9 & 6 & 0 & 2 & 5 & 7 & 3 \end{pmatrix},$$

or in "cycle notation" $(0,8,7,5)(1)(2,4,6)(3,9)$.

We summarise a stream cipher in the following definition.

Definition: A *stream cipher* over an alphabet of q symbols $a_1,\ldots,a_q$ requires a *key*, a random or pseudo-random string of symbols from the alphabet with the same length as the plaintext, and a *substitution table*, a Latin square of order q (whose entries are symbols from the alphabet, and whose rows and columns are indexed by these symbols). If the plaintext is $p_1p_2\ldots p_n$ and the key is $k_1k_2\ldots k_n$, then the ciphertext is $z_1z_2\ldots z_n$, where $z_t = p_t \oplus k_t$ for $t = 1,\ldots,n$; the operation $\oplus$ is defined as follows:

> $a_i \oplus a_j = a_k$ if and only if the symbol in the row labelled a_i and the column labelled a_j of the substitution table is a_k.

We extend the definition of $\oplus$ to denote this coordinate-wise operation on strings: thus, we write $z = p \oplus k$, where p,k,z are the plaintext, key, and ciphertext strings.

We also define the operation $\ominus$ by the rule that $p = z \ominus k$ if $z = p \oplus k$; thus, $\ominus$ describes the operation of decryption.

3.3 Fish

A simple improvement of the Vigenère cipher is to encipher twice using two different keys k_1 and k_2. Because of the additive nature of the cipher, this is the same as enciphering with $k_1 + k_2$. The advantage is that the length of the new key is the least common multiple of the lengths of k_1 and k_2. For example, if we encrypt a message once with the key `FOXES` and again with the key `WOLVES`, the new key

is obtained by encrypting a six-fold repeat of FOXES with a five-fold repeat of WOLVES, namely

BCIZWXKLPNJGTSDASPAGQJBWOTZSIK

The new key has period 30. Re-encrypting with a word of length 7 such as JAGUARS would have the effect that the new key has period 210.

This idea was exploited in the Second World War German cipher codenamed "Fish", so-called because it used the Siemens T52 machine known as *Sägefisch* (sawfish). This cipher, which was broken by the Bletchley Park cryptanalysts, is less well-known than the Enigma cipher, but is probably of even greater significance, since it was used for strategic messages, troop dispositions, etc., between the German High Command and the theatres of war. The book *Code Breakers: The Inside Story of Bletchley Park* (edited by F. H. Hinsley and Alan Stripp) gives more detail about breaking this cipher, which has been described as the greatest intellectual achievement of the war.

The Fish cipher employed the 5-bit International Telegraph Code, described earlier in this chapter. The five bits of each character in the plaintext were separated into five bitstreams which were enciphered separately and then reassembled into a sequnce of 5-bit words for transmission.

The encryption of each substream was by means of a stream cipher, generated by a mechanical device. The first stage consisted of one Vigenère cipher for each substream; the periods of these ciphers were 41, 31, 29, 26 and 23. Each cipher was implemented by a toothed wheel; the teeth could be extended or retracted, corresponding to a 1 or a 0 in the corresponding keyword. The wheels advanced one place after encrypting one bit from each stream in parallel. This was followed by a second cipher, like a Vigenère cipher but where we sometimes advance to the next letter of the keyword and sometimes remain with the same one, depending on the operation of two further wheels. The periods of the second ciphers were 43, 47, 51, 53 and 59, while the control wheels had periods 37 and 61. (The precise method of operation, and a diagram of the machine, appear in the book *Code Breakers*.) Figure 3.1 shows a diagram of the machine, from Tony Sale's "Codes and Ciphers" web page.

Since the wheel sizes are pairwise coprime, the period of the keystring generated by such a cipher is their product:

$$23 \cdots 61 = 16\,033\,955\,073\,056\,318\,658.$$

The keys of the different Vigenère ciphers and the control wheels could be set, but the lengths of the wheels was fixed.

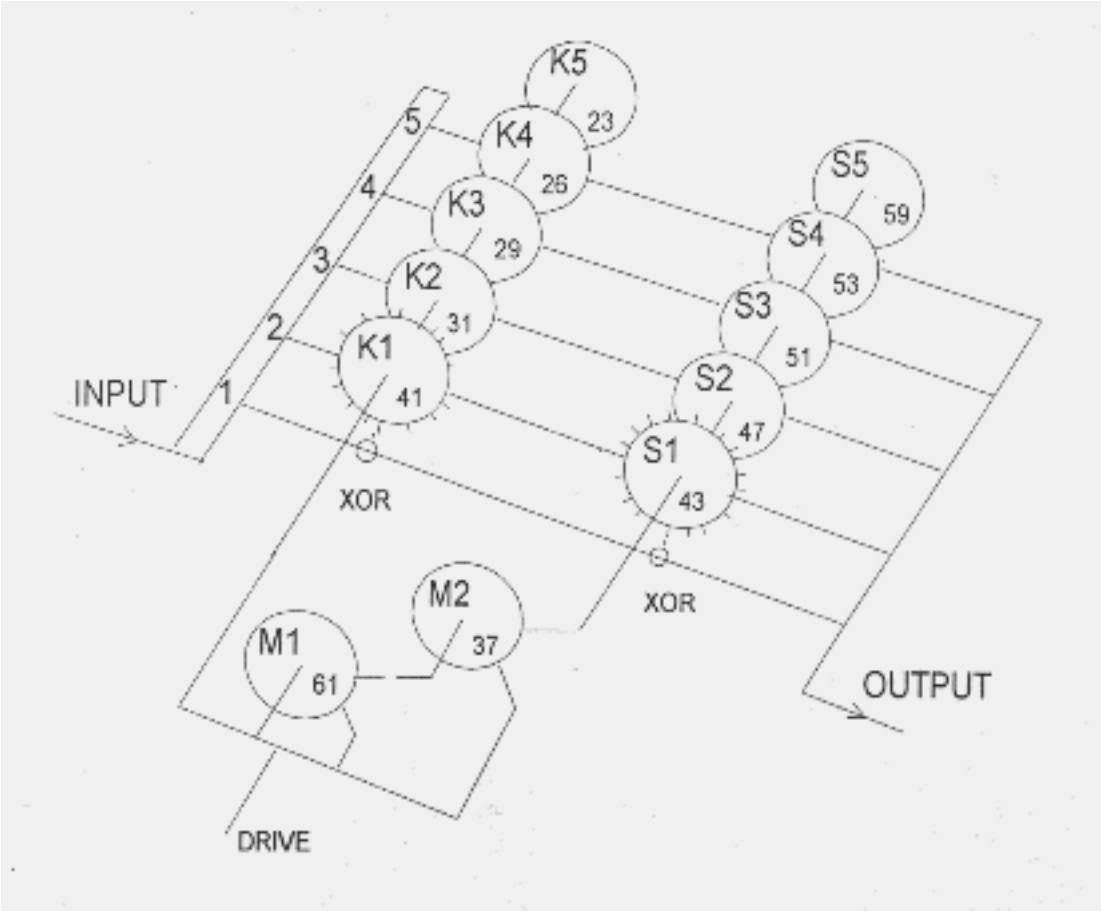


Figure 3.1: The Sägefisch cipher machine

It would have been possible for the Bletchley Park cryptanalysts to have assembled models of the cipher machines. But they felt that the supply of parts for such machines would have drawn attention to the fact that they were attempting to break the cipher. So instead they built electronic machines (including Colossus, the first stored-program computer) out of readily available parts used for telephone switchgear. This move from mechanical to electronic methods in cryptography was probably the most significant result of the Bletchley Park codebreakers.

3.4 One-time pads

A one-time pad is a stream cipher whose key is a random sequence of symbols from the alphabet. This means that, if the size of the alphabet is q and the length of the key is n, then each of the q^n keystrings has probability $1/q^n$. Said another way, each alphabet symbol is equally likely to appear in each position, and the symbols in the various positions are mutually independent. You may want to revise some elementary probability theory at this point (especially conditional probability and random variables).

Theorem 3.1 *A one-time pad is secure against statistical attack.*

Before we can prove the theorem, we have to say what it means. Before we receive the message, we have some prior estimate of the probabilities of various messages that might be sent (based perhaps on the statistics of language, perhaps on what we think that Alice might be saying to Bob). Let $p = p_0$ denote the event that the plaintext string is the particular string p_0. (Here we are thinking of p as a random variable and p_0 as a particular value that it might take.) Thus, probabilities $P(p = p_0)$ are assumed.

After we have intercepted a particular ciphertext string z_0, our new estimate of the probability is the conditional probability $P(p = p_0 \mid z = z_0)$. For example, if we can decrypt the cipher and determine that the plaintext sent was p_1, then $P(p = p_1 \mid z = z_0) = 1$, while $P(p = p_i \mid z = z_0) = 0$ if $p_i \neq p_1$. This represents the state where we have gained the maximum amount of information. A weaker requirement is just that our estimates of the probabilities of the various plaintexts have been changed by knowledge of the ciphertext.

Now Shannon's Theorem asserts that, if the key is random, then

$$P(p = p_0 \mid z = z_0) = P(p = p_0)$$

for any plaintext p_0. Thus, not only is it true that we cannot decrypt the message, but we cannot get any more information at all!

Let us prove this. By definition,

$$P(p = p_0 \mid z = z_0) = \frac{P(p = p_0 \text{ and } z = z_0)}{P(z = z_0)}.$$

Now the event $p = p_0$ and $z = z_0$ is the same as the event $p = p_0$ and $k = k_0$, where k denotes the key, and $p_0 \oplus k_0 = z_0$. (Any two of the plaintext, key and ciphertext uniquely determines the third.) Now the plaintext and the key are obviously

independent, so we have $P(p = p_0 \text{ and } k = k_0) = P(p = p_0) \cdot P(k = k_0) = P(p = p_0)/q^n$, where q is the alphabet size and n the length of the strings.

Let us compute $P(z = z_0)$. The ciphertext z_0 can arise in many ways, from any plaintext p_i and key k_i satisfying $p_i \oplus k_i = z_0$. These events are pairwise disjoint. So, is $\sum$ denotes the sum over all such pairs (p_i, k_i), we have

$$\begin{aligned} P(z = z_0) &= \sum P(p = p_i) \cdot P(k = k_i) \\ &= \sum P(p = p_i)/q^n \\ &= 1/q^n. \end{aligned}$$

Here the first equation holds because of the assumption that the keys are random, and the second just says that the prior probabilities of the various plaintexts must add up to 1.

Finally, we get

$$P(p = p_0 \mid z = z_0) = \frac{P(p = p_0)/q^n}{1/q^n} = P(p = p_0),$$

and the proof is complete.

In fact an even stronger property holds. If we already know the decryption of part of the ciphertext, then clearly this will alter our estimated probabilities for the rest of the text. However, knowledge of the ciphertext does not give any further information! We will see that, for a widely used class of stream ciphers (those based on shift registers), this assumption is far from true: knowledge of the ciphertext and a small amount of plaintext enables the cipher to be broken completely.

3.5 Golomb's Postulates

How do we tell if a sequence is random?

This is a very deep question, and several different solutions have been proposed. By definition, 'random' means 'selected from the set of all possible sequences, any sequence being equally likely', or (what amounts to the same thing, 'the symbols in the string are chosen independently with equal probability'. But this definition refers to the set of all possible sequences, and doesn't tell us anything about a single sequence. Indeed, any sequence can occur, even a constant sequence!

A completely different definition was proposed by Kolmogorov, who said:

> A sequence is random if it cannot be generated by an algorithm with a short description (i.e. much shorter than the sequence itself).

Using this definition, the keystring of the Fish cipher, or the string of digits of π, is not random. However, the definition is not easy to apply.

A more practical test was given by Golomb, who proposed three postulates. To state Golomb's postulates, we need a couple of definitions. Suppose that $a = a_0a_1\ldots a_{n-1}$ is a binary sequence. We regard it as cyclic, so that a_0 is regarded as following a_{n-1}. A *run* in the sequence is a subsequence such that all the entries are the same, which is as long as possible: that is, either a row of 1s with 0s at each end, or a row of 0s with 1s at each end. The *correlation* of two sequences a and b is defined to be $\sum a_ib_i$. The correlation of the sequence a with a cyclic shift of itself is called an *autocorrelation* of a; it is *in phase* if the shift is zero, and *out of phase* otherwise. Thus, the autocorrelation is $\sum a_ia_{i+m}$, where the subscripts are mod n; it is in phase if $m = 0$ and out of phase otherwise. (Sometimes in the literature a renormalisation is applied to the correlation; this doesn't affect the postulates below.)

Golomb's postulates are the following:

(G1) The numbers of 0s and 1s in the sequence are as near as possible to $n/2$ (that is, exactly $n/2$ if $n/2$ is even, and $(n\pm1)/2$ if n is odd).

(G2) The number of runs of given length should halve when the length is increased by one (as long as possible), and where possible equally many runs of given length should consist of 0s as of 1s.

(G3) The out-of-phase autocorrelation should be constant (independent of the shift).

A sequence satisfying these postulates is called a *pseudo-noise sequence* or *PN-sequence*.

For example, consider the sequence

000100110101111

which we regard as being continued for ever in cyclic fashion. There are seven 1s and eight 0s, so (G1) is true. The runs are as follows:

- four of length 1, two 0s (beginning at positions 8 and 10) and two 1s (beginning at 3 and 9);

- two of length 2, one 00 (beginning at 4) and one 11 (beginning at 6);
- one of length 3, 000 beginning at 0;
- one of length 4, 1111 beginning at 11.

So (G2) is satisfied. For (G3), compare the sequence with each of its cyclic shifts:

000100110101111
100010011010111
110001001101011
111000100110101
111100010011010
011110001001101
101111000100110
010111100010011
101011110001001
110101111000100
011010111100010
001101011110001
100110101111000
010011010111100
001001101011110

We see by inspection that the autocorrelation of any two rows is equal to 4. Of course the in-phase autocorrelation is 8.

Exercise: Write down a string of 'random' bits, say of length 32. (That is, try to avoid any obvious patterns.) How close does your string come to satisfying Golomb's postulates?

Now toss a coin 32 times to generate random bits. Does this string fit Golomb's postulates better?

3.6 Shift registers

One method which has been widely used for generating pseudo-random binary sequences involves shift registers.

Figure 3.2 shows a shift register.

Each of the boxes in the shift register contains one bit (zero or one). The shift register is controlled by a clock which ticks at discrete time intervals. When the clock ticks, the contents x_0 and x_1 of the first two boxes are added (mod 2); then

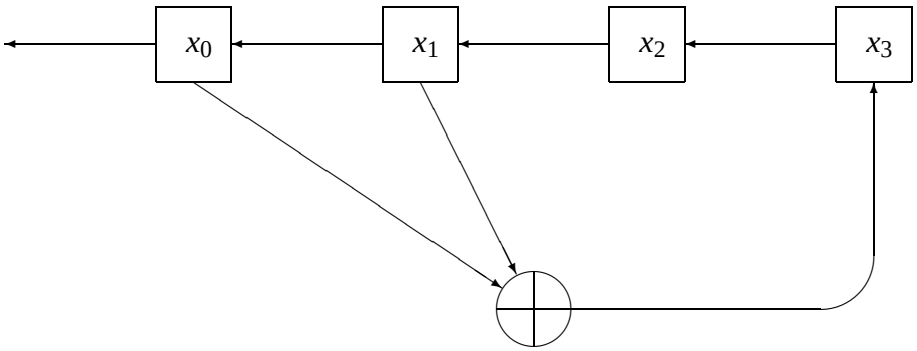


Figure 3.2: A shift register

the contents of each box is shifted one place left (that of the first box is output) and the result of the addition is put in the last box.

Suppose that the boxes initially contain 0001. Then, at successive clock ticks, they become 0010, 0100, 1001, 0011, 0110, 1101, 1010, 0101, 1011, 0111, 1111, 1110, 1100, 1000, 0001, and the machine outputs the sequence

000100110101111

At this point, the contents have returned to their original values, and the machine then repeats the same cycle indefinitely.

We see that, for this particular shift register, every possible binary 4-tuple except 0000 occurs precisely once in a cycle as the contents of the boxes. Moreover, the contents of the boxes at stage n become the next four bits of the output string. So, if we consider the string as continuing indefinitely, and if we look at it through a window which shows just four bits at a time, then we see each of the $2^4 - 1 = 15$ non-zero 4-tuples just once in each cycle. Note that we could start with any non-zero 4-tuple and the same cycle would be obtained.

On the other hand, if we start with 0 in each box, then the contents of the boxes will always be 0, and the output string consists entirely of zeros – not very good as a pseudo-random string.

In general, a shift register works in the same way. It is specified by giving

(a) the number of boxes;

(b) which boxes are connected to the “adder”.

If there are n boxes, we speak of an n-bit shift register. Its configuration at any given time is the binary n-tuple giving the contents of the boxes at that time.

For reasons that will become clear in the next section, it is convenient to describe a shift register by a polynomial over the binary field. The degree n of the polynomial is the number of boxes; the coefficient of x^i is 1 if $i = n$ or if the ith box is connected to the adder, and 0 otherwise. (We number the boxes from 0 on the left to $n-1$ on the right.) Thus, the polynomial describing the shift register in Figure 3.2 is

$$x^4 + x + 1.$$

Proposition 3.2 *Suppose that a shift register is described by the polynomial*

$$x^n + \sum_{i=0}^{n-1} a_i x^i.$$

Then its output sequence is given by the recurrence relation

$$x_{k+n} = \sum_{i=0}^{n-1} a_i x_{k+i}.$$

Proof: Suppose that the configuration is $(u_0, \ldots, u_{n-1})$. At the next clock tick, the adder computes $t = \sum_{i=0}^{n-1} a_i u_i$. The next n bits output are, in order, $x_k = u_0$, $x_{k+1} = u_1$, ..., $x_{k+n-1} = u_{n-1}$, $x_{k+n} = t$. Hence the sequence is given by the recurrence relation.

An n-bit shift register (one with n boxes $x_0, \ldots, x_{n-1}$) which starts in a non-zero configuration must return to its starting point in at most $2^n - 1$ steps, since there are exactly this many non-zero configurations it can have. Thus, its period is at most $2^n - 1$. An n-bit shift register is said to be *primitive* if is period is $2^n - 1$; that is, if it has the property that, if the starting configuration is non-zero, then each of the $2^n - 1$ non-zero n-tuples occurs once as a configuration in the course of a cycle. The next theorem asserts that primitive shift registers exist with any given number of bits.

Theorem 3.3 *For any positive integer n, there is a primitive n-bit shift register.*

Of course, it is easy to construct a shift register with a moderate number of bits, say 30 or 100. We can ensure (by choosing a primitive shift register) that its output sequence will not repeat during the lifetime of the universe!

Algebraic formulation

The behaviour of the shift register can be described algebraically. If $x = (x_0, x_1, x_2, x_3)$ are the contents of the shift register at any moment, and $y = (y_0, y_1, y_2, y_3)$ the contents after the clock ticks, then we have

$$\begin{array}{lcccccc} y_0 & = & & & x_1 & & \\ y_1 & = & & & & x_2 & \\ y_2 & = & & & & & x_3 \\ y_3 & = & x_0 & + & x_1 & & \end{array}$$

or, in matrix terms, $y' = Ax'$, where A is the matrix

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

(Here x' is the transpose of x, the column vector corresponding to the row vector x.)

The matrix A satisfies $A^{15} = I$, and no smaller power of A is equal to I. If V denotes the 4-dimensional vector space over the binary field, then for any non-zero vector $x \in V$, the fifteen vectors

$$x', Ax', A^2x', \ldots, A^{14}x'$$

are distinct and comprise all the non-zero vectors in V.

The connection between the polynomial and the matrix is simple:

> The polynomial of a shift register is equal to the characteristic polynomial (and to the minimal polynomial) of its matrix.

For, given a polynomial $f(x) = x^n + a_{n-1}x^{n-1} + \cdots + a_1x + a_0$, the *companion matrix* of f is defined to be the matrix

$$C(f) = \begin{pmatrix} 0 & 1 & 0 & \ldots & 0 & 0 \\ 0 & 0 & 1 & 0 & \ldots & 0 \\ \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\ 0 & 0 & 0 & \ldots & 0 & 1 \\ -a_0 & -a_1 & -a_2 & \ldots & -a_{n-2} & -a_{n-1} \end{pmatrix}$$

with zeros everywhere except for ones above the diagonal and the coefficients of f in reverse order with the sign changed in the bottom row. It is a standard result that the characteristic and minimal polynomials of $C(f)$ are both equal to f. Now over the binary field, $-a$ is the same as a, and the matrix associated with the shift register is precisely $C(f)$, so has the same characteristic and minimal polynomials.

We call a polynomial of degree n *primitive* if its associated shift register is primitive. Now the following theorem holds:

Theorem 3.4 *A primitive polynomial is irreducible.*

The proof of this theorem depends on the theory of finite fields and is beyond the scope of the course.

Example: Suppose that $n = 4$. How do we find all the primitive polynomials? First we find the irreducible polynomials. Let

$$f(x) = x^4 + ax_3 + bx^2 + cx + d$$

be a polynomial over $\mathbb{Z}/(2)$, so that all the coefficients are 0 or 1. There are $2^4 = 16$ polynomials altogether. Now, by the remainder theorem, if $f(0) = 0$, that is, $d = 0$, then x is a factor of $f(x)$; and if $f(1) = 0$, that is, $1 + a + b + c + d = 0$, then $x - 1$ is a factor (note that $x - 1$ is the same as $x + 1$. So we must have $d = 1$ and $a + b + c = 1$. Of the sixteen polynomials, just four pass these tests, namely

$$x^4 + x + 1, \quad x^4 + x^2 + 1, \quad x^4 + x^3 + 1, \quad x^4 + x^3 + x^2 + x + 1.$$

Now there is an irreducible polynomial of degree 2, namely $x^2 + x + 1$, and

$$(x^2 + x + 1)^2 = x^4 + x^2 + 1$$

is reducible. This leaves three polynomials. All of them are irreducible, since we have exhausted all the possible factorisations.

Now $x^4 + x + 1$ is primitive; this is the polynomial of the shift register with which we started. Similarly it can be checked that $x^4 + x^3 + 1$ is primitive. However, if we take the polynomial $x^4 + x^3 + x^2 + x + 1$ (with corresponding recurrence relation $x_{i+4} = x_{i+3} + x_{i+2} + x_{i+1} + x_i$), the starting configuration 0001 generates the sequence

$$000110001100011\ldots$$

of period 5. The other starting configurations also produce output of period 5. This polynomial is not primitive.

Stream ciphers from shift registers

The sequences generated by shift registers are not of course random. In Kolmogorov's sense, they are very far from being random, since they are generated by a very simple machine. However, they are pseudo-noise sequences in Golomb's sense:

Theorem 3.5 *The output sequence of any primitive shift register satisfies Golomb's postulates.*

It is easy to see that postulate (G1) is satisfied. Remember that every non-zero n-tuple occurs exactly once as the configuration of the shift register in the course of the cycle. Now of the $2^n - 1$ possible non-zero n-tuples, $2^{n-1} - 1$ begin with zero and 2^{n-1} with one; so the cycle contains $2^{n-1} - 1$ zeros and 2^{n-1} ones, in accordance with (G1).

We will not prove all of the theorem here; the proof uses the theory of finite fields. In fact, the string of length 15 which we used in the preceding chapter is the output of the shift register with which we began this chapter.

Breaking a shift register

Although primitive shift registers have many good properties, such as satisfying Golomb's postulates, they have one fatal flaw: it doesn't take much information to break a stream cipher based on a shift register.

Theorem 3.6 *Suppose that a stream cipher is based on an n-bit shift register. Suppose that $2n$ consecutive bits of ciphertext and the corresponding plaintext are known. Then the cipher can be broken.*

Proof: From the $2n$ bits of ciphertext and corresponding plaintext, we obtain $2n$ consecutive bits of the keystring, say $u_0, u_1, \ldots, u_{2n-1}$. From Proposition 3.2, we have

$$\begin{aligned} u_n &= a_0u_0 + a_1u_1 + \cdots + a_{n-1}u_{n-1}, \\ u_{n+1} &= a_0u_1 + a_1u_2 + \cdots + a_{n-1}u_n, \\ &\cdots \\ u_{2n-1} &= a_0u_{n-1} + a_1u_n + \cdots + a_{n-1}u_{2n-2} \end{aligned}$$

This looks like a set of linear equations for the us, with the as as coefficients. But remember that in this case we know the us but not the as. So we regard them as equations for the unknowns $a_0, \ldots, a_{n-1}$. There are equally many equations as unknowns (namely n), and it is possible to show that the equations have a unique solution.

Thus we can determine the shift register, and then simulate its action (starting with the configuration $(u_0, \ldots, u_{n-1})$ to find the entire keystring.

The moral of the story is that any device that produces a long-period sequence from a small amount of data is vulnerable.

Example: Suppose that 11010110 is part of the output of a 4-bit shift register. We obtain the equations

$$\begin{array}{rcccccccc} 0 & = & a_0 & + & a_1 & & & + & a_3, \\ 1 & = & a_0 & & & + & a_2, & & \\ 1 & = & & & a_1 & & & + & a_3, \\ 0 & = & a_0 & & & + & a_2 & + & a_3. \end{array}$$

These equations have solution $a_0 = 1$, $a_1 = 0$, $a_2 = 0$, $a_3 = 1$. So the shift register has polynomial $x^4 + x^3 + 1$, and a period of its output is

$$1101011001000111$$

We see that the shift register is primitive.

How could $2n$ bits of plaintext be obtained? There are a number of methods. First of all, by guesswork. If Alice always starts her letters with "Dear Bob," we can make use of this fact. Another method would be to physically steal the plaintext from either Alice or Bob.

The breaking of the Fish cipher illustrates how Alice's carelessness can help Eve. The first step that led to the breaking of the Fish cipher occurred when the cryptanalysts discovered that two long messages had been enciphered using the same key (that is, the same settings and initial state of the wheels). Thus, we have

$$z = p \oplus k, \qquad z' = p' \oplus k,$$

where $\oplus$ here denotes bitwise binary addition. From the properties of binary addition, we deduce that

$$z \oplus z' = p \oplus p'.$$

This means that, when the two ciphertexts are added, the key dieappears, and we have the sum of two plaintexts. Now these can be teased apart by frequency analysis, to find the two plaintexts p and p'. Now we can find the key $k = p \oplus z$. The cryptanalysts used the key to deduce the structure of the cipher machine. This is similar to (but rather more complicated than) our use of $2n$ bits of key to break an n-bit shift register.

Worked example The *seven-bit ASCII code* represents letters, digits, and punctuation as characters from the set of integers in the range $32 \ldots 127$; the capital letters `A...Z` are represented by $65 \ldots 90$, and lower-case letters `a...z` by $97 \ldots 112$. Integers in the range $0 \ldots 31$ are used for control codes. The integers are then written in base 2, as 7-tuples of zeros and ones.

You intercept the string

00001101101110101011111101110100110111100100111100001011000101010101010

You have reason to believe that it is a message in seven-bit ASCII encrypted by means of a stream cipher based on a seven-bit shift register, and that the first two letters of the message are `Su`. Decrypt the string.

Solution The 7-bit ASCII code for `Su` is 10100111110101. Subtracting these fourteen bits of plaintext from the first fourteen bits of ciphertext gives us fourteen bits of key: 10101010011011. So the equations for the shift register are

$$\begin{array}{cccccccccccccc}
0 & = & a_0 & & + & a_2 & & + & a_4 & & & + & a_6 \\
0 & = & & a_1 & & & + & a_3 & & + & a_5 & & \\
1 & = & a_0 & & + & a_2 & & + & a_4 & & & & \\
1 & = & & a_1 & & & + & a_3 & & & & + & a_6 \\
0 & = & a_0 & & + & a_2 & & & & + & a_5 & + & a_6 \\
1 & = & & a_1 & & & & + & a_4 & + & a_5 & & \\
1 & = & a_0 & & & & + & a_3 & + & a_4 & & + & a_6
\end{array}$$

Solving, we find $(a_0, \ldots, a_6) = (1,1,0,1,0,0,1)$, so the shift register polynomial is $x^7 + x^6 + x^3 + x + 1$. Now we can continue the key to 70 bits using the recurrence relation $x_{n+7} = x_{n+6} + x_{n+3} + x_{n+1} + x_n$ and subtract it from the ciphertext to obtain the plaintext, and then divide the plaintext into 7-bit blocks and decode each block to obtain the message: `Surrender!`

3.7 Finite fields

Any serious investigation of shift registers must observe that they are very closely connected with finite fields. A field is a set with two operations (addition and multiplication) in which the 'usual rules' apply. For example, the rational, real or complex numbers, or the integers modulo p (where p is prime) are fields.

The finite fields were classified by Galois around 1830:

Theorem 3.7 *The order of a finite field must be a prime power. For every prime power q, there is a field with q elements, and it is unique up to isomorphism.*

The field with q elements is denoted by $\mathrm{GF}(q)$ (for 'Galois field') in honour of Galois.

Two properties of finite fields are important here:

Theorem 3.8 *The multiplicative group of a finite field is cyclic.*

This means that $\mathrm{GF}(q)$ contains an element α with the property that all the $q-1$ non-zero elements are powers of α. Thus, $\alpha^{q-1}=1$, but no smaller power of α is equal to 1. Such an element α is said to be a *primitive element* of $\mathrm{GF}(q)$. The number of primitive elements of $\mathrm{GF}(q)$ is equal to $\phi(q-1)$, where ϕ is Euler's function.

Theorem 3.9 *Let p and p_1 be primes. The field $\mathrm{GF}(p^n)$ contains a subfield $\mathrm{GF}(p_1^m)$ if and only if $p=p_1$ and m divides n. In this case, there is a unique subfield $\mathrm{GF}(p^m)$ of $\mathrm{GF}(p^n)$.*

Now let q be a given prime power. The field $\mathrm{GF}(q^n)$ contains a unique subfield $\mathrm{GF}(q)$. For each element $\theta \in \mathrm{GF}(q^n)$, there is a *minimal polynomial* of θ over $\mathrm{GF}(q)$, that is, a monic polynomial satisfied by θ. This polynomial is always irreducible, and its degree is equal to m if the smallest subfield of $\mathrm{GF}(q^n)$ containing $\mathrm{GF}(q)$ and θ is $\mathrm{GF}(q^m)$.

The monic polynomial of θ has degree n if and only if θ lies in no subfield of $\mathrm{GF}(q^n)$ containing $\mathrm{GF}(q)$ (except $\mathrm{GF}(q^n)$ itself). Every irreducible polynomial of degree n over $\mathrm{GF}(q)$ is the minimal polynomial of exactly n elements of $\mathrm{GF}(q^n)$.

Now consider the case where $q=2$. We begin by reversing the procedure and constructing $\mathrm{GF}(2^4)$ as an example. Let α be a root of the irreducible polynomial

x^4+x+1 over GF(2). Thus, $\alpha^4+\alpha+1=0$, or (since $-1=+1$) $\alpha^4=\alpha+1$. We can make a table of powers of α as follows:

$$\begin{array}{lcccccccc}
\alpha^0 &=& & & & & & & 1\\
\alpha^1 &=& & & & & \alpha & & \\
\alpha^2 &=& & & \alpha^2 & & & & \\
\alpha^3 &=& \alpha^3 & & & & & & \\
\alpha^4 &=& & & & & \alpha & + & 1\\
\alpha^5 &=& & & \alpha^2 & + & \alpha & & \\
\alpha^6 &=& \alpha^3 & + & \alpha^2 & & & & \\
\alpha^7 &=& \alpha^3 & & & + & \alpha & + & 1\\
\alpha^8 &=& & & \alpha^2 & & & + & 1\\
\alpha^9 &=& \alpha^3 & & & +\alpha & & & \\
\alpha^{10} &=& & & \alpha^2 & + & \alpha & + & 1\\
\alpha^{11} &=& \alpha^3 & + & \alpha^2 & + & \alpha & & \\
\alpha^{12} &=& \alpha^3 & + & \alpha^2 & + & \alpha & + & 1\\
\alpha^{13} &=& \alpha^3 & + & \alpha^2 & & & + & 1\\
\alpha^{14} &=& \alpha^3 & & & & & + & 1
\end{array}$$

and $\alpha^{15}=1=\alpha^0$, so the sequence repeats (like the shift register). We see that α is a primitive element of the field GF(2^4); the field consists of zero and the fifteen powers of α.

Using this table as a table of logarithms, we can do arithmetic in the field. For example,

$$\begin{aligned}
(\alpha^2+\alpha+1)+(\alpha^3+\alpha^2+\alpha) &= \alpha^3+1,\\
(\alpha^2+\alpha+1)\cdot(\alpha^3+\alpha^2+\alpha) &= \alpha^{10}\cdot\alpha 11=\alpha^6=\alpha^3+\alpha^2.
\end{aligned}$$

Now let $\beta=\alpha^7$. We have

$$\begin{aligned}
\beta^2 &= \alpha^{14}=\alpha^3+1,\\
\beta^3 &= \alpha^6=\alpha^3+\alpha^2,\\
\beta^4 &= \alpha^{13}=\alpha^3+\alpha^2+1.
\end{aligned}$$

So we see that $\beta^4=\beta^3+1$, so that β satisfies the primitive polynomial x^4+x^3+1.

Similarly we find that $\gamma=\alpha^3$ satisfies the irreducible but not primitive polynomial $x^4+x^3+x^2+x+1$, while $\delta=\alpha^5$ has minimal polynomial x^2+x+1 and lies in a subfield GF(4) consisting of the elements $0,1,\alpha^5,\alpha^{10}$.

The three irreducible polynomials of degree 4 each have four roots. The irreducible polynomial x^2+x+1 has two roots. The two elements $0,1$ have minimal polynomials x and $x+1$ respectively of degree 1. Thus all elements of $\mathrm{GF}(16)$ are accounted for.

Theorem 3.10 *Let θ be an element of $\mathrm{GF}(2^n)$ with minimal polynomial $f(x)$ of degree n. Then $f(x)$ is a primitive polynomial (in the sense that the associated shift register has period 2^n-1) if and only if θ is a primitive element of $\mathrm{GF}(2^n)$.*

For example, suppose that $n=4$. The proper subfields of $\mathrm{GF}(16)$ are

$$\mathrm{GF}(2) \subseteq \mathrm{GF}(4) \subseteq \mathrm{GF}(16),$$

where $\mathrm{GF}(2)$ is the binary field $\mathbb{Z}/(2)$. So there are 12 elements of $\mathrm{GF}(16)$ which lie in no proper subfield, and thus $12/4=3$ irreducible polynomials of degree 4. Moreover, there are $\phi(15)=2\cdot 4=8$ primitive elements of $\mathrm{GF}(16)$, and hence $8/4=2$ primitive polynomials. These agree with what we found by hand earlier.

3.8 Latin squares

Why do we need a Latin square for the substitution table in a stream cipher?

In the article "Japanese Army Air Force Codes at Bletchley Park and Delhi", by Alan Stripp, in the book *Code Breakers: The Inside Story of Bletchley Park* (edited by F. H. Hinsley and Alan Stripp), the following example is given of a substitution table supposedly used in the Japanese Army Air Force cipher J6633 (Figure 3.4).

By inspection, it is not a Latin square. It fails in various ways; for example,

(a) symbol 0 occurs twice in column 4 (in rows 2 and 6);

(b) symbol 1 occurs twice in row 1 (in columns 6 and 8).

The consequences of these two flaws are quite different.

Having a repeated element in a column means that the column is not a permutation of the alphabet, and so we cannot use the key to decrypt unambiguously. If the ciphertext letter was 0 and the corresponding key letter was 4, we wouldn't know whether the plaintext letter was 2 or 6.

Having a repeated element in a row does not stop us from decrypting the message. But it destroys the randomness of the key, and gives the cryptanalyst a small

	0	1	2	3	4	5	6	7	8	9
0	4	9	5	3	2	7	0	1	6	8
1	7	5	0	9	3	2	1	8	1	4
2	3	1	7	2	8	0	9	6	9	7
3	0	8	4	7	0	1	3	4	5	2
4	5	3	2	4	9	3	8	2	7	6
5	9	0	1	6	7	5	4	7	2	3
6	2	6	8	0	0	9	7	5	3	1
7	6	2	6	1	4	8	6	0	8	5
8	1	7	9	7	1	4	5	9	0	7
9	8	4	3	5	5	6	2	3	4	0

Table 3.4: Japanese Army Air Force cipher J6633

amount of leverage: the ciphertext string now carries a small amount of information about the plaintext. For example, suppose that we are using the square in Figure 3.4. If the ciphertext symbol 0 is received, Eve can be sure that the plaintext is *not* 4, since 0 doesn't occur in the fourth row of the table.

To take this to extremes, suppose that we used a substitution square in which the columns were permutations but all rows were constant, say

	0	1	2	3	4	5	6	7	8	9
0	4	4	4	4	4	4	4	4	4	4
1	7	7	7	7	7	7	7	7	7	7
2	3	3	3	3	3	3	3	3	3	3
3	0	0	0	0	0	0	0	0	0	0
4	5	5	5	5	5	5	5	5	5	5
5	9	9	9	9	9	9	9	9	9	9
6	2	2	2	2	2	2	2	2	2	2
7	6	6	6	6	6	6	6	6	6	6
8	1	1	1	1	1	1	1	1	1	1
9	8	8	8	8	8	8	8	8	8	8

In this case, the plaintext letter 0 is always replaced by the ciphertext letter 4, regardless of the key. In other words, this is a simple substitution cipher, and the key is irrelevant. It can be broken by standard frequency analysis. The same general principle applies even if rows are not constant, as the next example shows.

Worked example A message in a 3-letter alphabet $\{1,2,3\}$ has been encrypted using a random keystring and the substitution table

	1	2	3
1	2	3	1
2	1	2	2
3	3	1	3

The message has length 3. Before intercepting the ciphertext, your estimates of the probabilities of plaintext strings are

$$P(112) = 0.1, \quad P(231) = 0.2, \quad P(332) = 0.3, \quad P(313) = 0.4,$$

and all other probabilities zero.

You intercept the ciphertext 132. Calculate the conditional probabilities of the plaintext strings given this information.

Does your answer contradict Shannon's Theorem?

Solution We follow the argument in the proof of Shannon's Theorem. First we have to decide which keys would encrypt each possible plaintext as the given ciphertext. We see that $112 \oplus k = 132$ holds for $k = 322$ or 323 (the ambiguity because of the two occurences of 2 in the second row of the table). So $P(z = 132 \mid p = 112) = 2/27$. Similarly, $231 \oplus k = 132$ holds for $k = 113$ or $k = 133$, giving $P(z = 132 \mid p = 231) = 2/27$; and $332 \oplus k = 132$ holds for $k = 212, 232, 213, 233$, so that $P(z = 132 \mid p = 332) = 4/27$. Finally, $313 \oplus k = 132$ is impossible, since 2 does not occur in the third row of the table; so $P(z = 132 \mid p = 313) = 0$.

The Theorem of Total Probability gives

$$P(z = 132) = \frac{2}{27} \cdot \frac{1}{10} + \frac{2}{27} \cdot \frac{2}{10} + \frac{4}{27} \cdot \frac{3}{10} + 0 \cdot \frac{4}{10} = \frac{18}{270}.$$

From Bayes Theorem we find

$$\begin{aligned}
P(p = 112 \mid z = 132) &= \frac{(2/27) \cdot (1/10)}{18/270} = \frac{1}{9}, \\
P(p = 231 \mid z = 132) &= \frac{(2/27) \cdot (2/10)}{18/270} = \frac{2}{9}, \\
P(p = 332 \mid z = 132) &= \frac{(4/27) \cdot (3/10)}{18/270} = \frac{2}{3}, \\
P(p = 313 \mid z = 132) &= 0.
\end{aligned}$$

These are not the same as the prior probabilities, so we have gained some information. However, Shannon's Theorem is not contradicted, since one of its hypotheses asserts that the substitution table is a Latin square, which is not true in this case.

Latin squares are very plentiful. Their first practical use was in experimental design in statistics, where they were introduced by R. A. Fisher. (He is commemorated in Caius College, Cambridge, by a stained glass Latin square in a window of the dining hall: see Figure 3.3.)

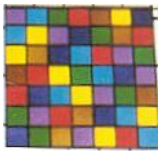


Figure 3.3: The R. A. Fisher window in Caius College, Cambridge

In the early days of the subject, it was recommended that randomization of the experiment should include choosing a random Latin square for the design. The only way this could be done was by tabulating all Latin squares of relatively small order, and choosing one at random from the tables. (The famous tables of Fisher and Yates include such lists.) Subsequently this practice was abandoned. Now, however, a Markov chain method for choosing a random Latin square has been proposed by Jacobson and Matthews.

Another feature of Latin squares is that we can construct them by building up row by row. For $k \leq n$, we define a $k \times n$ *Latin rectangle* to be an array with entries from the set $\{1, \ldots, n\}$ such that each symbol occurs once in each row and at most once in each column. Now any $k \times n$ Latin rectangle with $k < n$ can be "completed" to a Latin square.

Self-inverse squares

Let $A = (a_{ij})$ be a Latin square of order n. We can construct three further squares from A as follows. Suppose that $a_{ij} = k$.

- $A^{(12)}$ has (j,i) entry k. (Thus $A^{(12)}$ is the transpose of A.)
- $A^{(13)}$ has (k,j) entry i. (This is sometimes called the *adjugate* of A.)

- $A^{(23)}$ has (i,k) entry j. (This is sometimes called the *conjugate* of A.)

The reason for the notation is as follows. We can completely describe a Latin square A by the list of n^2 triples (i,j,k) for which the (i,j) entry of the square is k. For example, the square

$$A = \begin{array}{|c|c|c|}\hline 1 & 2 & 3 \\ \hline 2 & 3 & 1 \\ \hline 3 & 1 & 2 \\ \hline \end{array}$$

would be given by the nine triples

$$(1,1,1),(1,2,2),(1,3,3),(2,1,2),(2,2,3),(2,3,1),(3,1,3),(3,2,1),(3,3,2).$$

These can be written as the columns of a 3×9 array:

$$\begin{pmatrix} 1 & 1 & 1 & 2 & 2 & 2 & 3 & 3 & 3 \\ 1 & 2 & 3 & 1 & 2 & 3 & 1 & 2 & 3 \\ 1 & 2 & 3 & 2 & 3 & 1 & 3 & 1 & 2 \end{pmatrix}.$$

Now the square $A^{(12)}$ is obtained by interchanging the first and second rows in this array; and similarly for the others. In the above example, $A^{(12)}$ is the same as A (as A is symmetric), while

$$A^{(23)} = \begin{array}{|c|c|c|}\hline 1 & 2 & 3 \\ \hline 3 & 1 & 2 \\ \hline 2 & 3 & 1 \\ \hline \end{array}.$$

Proposition 3.11 *If A is a Latin square, so are $A^{(12)}$, $A^{(13)}$, and $A^{(23)}$.*

This holds because, when we represent a Latin square as a set of triples as above, then a triple is uniquely determined by any two of its elements. (This means that specifying any two of the row, column and entry determines the other.) This property is still satisfied if we permute the entries.

If A is the substitution square used for encryption with a stream cipher, then $A^{(13)}$ is the substitution square used for decryption of the same cipher: this follows immediately from the definitions. Hence we will call this square the *inverse* of A.

A Latin square is called *self-inverse* if it is equal to its inverse square. If this property holds, then we have the simplification that the same square is used for both encryption and decryption.

The Vigenère square A, whose (i,j) entry is $i+j \bmod q$, is not in general self-inverse. However, if we take the (i,j) entry to be $j-i \bmod q$, we do obtain

a self-inverse Latin square: for, if $j - i \equiv k \pmod{q}$, then $j - k \equiv i \pmod{q}$. (This is actually the square $A^{(23)}$.) For $q = 4$, the subtraction square is

0	1	2	3
3	0	1	2
2	3	0	1
1	2	3	0

In the case $q = 2$, of course, subtraction is the same as addition, and the Vigenère square is self-inverse.

There are many other self-inverse Latin squares apart from the subtraction square.

The 3×9 array we constructed above from a Latin square is a particular case of an *orthogonal array*. We will see later how such arrays are used in secret sharing schemes.

3.9 Entropy

The concept of entropy originated in nineteenth-century thermodynamics as a measure of the disorder of a complicated physical system. Shannon introduced it into information theory, where it provides a very convenient measure of information. The background probability theory can be found in any book on the subject, or in the *Notes on Probability* on the Web.

Let X be a random variable on a probability space $\mathcal{S}$ with probability function P. (Recall that this simply means that X is a function on $\mathcal{S}$. In elementary probability theory we assume that the values of X are numbers, but they can be anything at all. Here we only consider finite probability spaces.) The entropy of X is a measure of our ignorance about the value of X (or, equivalently, the amount of information we would gain if we performed an observation and learned the value of X). This interpretation suggests that the entropy of X should be zero if X is constant (since then measuring X will tell us nothing we don't already know) and maximum if all the values of X have the same probability.

The definition is as follows. The *entropy* of X is given by the formula

$$H(X) = \sum_{i=1}^{n} \Pr(X = x_i) \log_2 \Pr(X = x_i),$$

where $x_1, \ldots, x_n$ are the possible values of X.

It is easily verified that X has the required properties:

Proposition 3.12 *1. $H(X) \geq 0$, with equality if and only if there is a value x such that $\Pr(X = x) = 1$.*

2. If X takes n values $x_1, \ldots, x_n$, then $H(X) \leq \log_2(n)$, with equality if and only if $\Pr(X = x_i) = 1/n$ for $i = 1, \ldots, n$.

Example Suppose that I toss a fair coin n times; the values of the random variable X are the 2^n possible bitstrings produced (where, say, heads $= 1$, tails $= 0$). Then $H(X) = \log_2 2^n = n$. That is, n random bits have entropy n. So the units of entropy are "bits"; observing a random variable X gives us "the same amount of information" as knowledge of $H(X)$ random bits.

If A is an event with non-zero probability, then the *conditional random variable* $X_A = X \mid A$ is defined by the rule that

$$\Pr(X_A = x_i) = \Pr(X = x_i \mid A) = \frac{\Pr(X = x_i \text{ and } A)}{\Pr(A)}.$$

The random variable $X \mid A$ now has entropy $H(X \mid A)$ according to the usual formula.

In particular, let X and Y be random variables. For each value y_j of Y, there is a conditional entropy $H(X \mid (Y = y_j))$. Then we define the conditional entropy of X given Y to be the weighted average (expected value) of $H(X \mid (Y = y_j))$; that is,

$$H(X \mid Y) = \sum_{j=1}^{m} H(X \mid (Y = y_j)) \Pr(Y = y_j),$$

where $y_1, \ldots, y_m$ are the values of Y.

A short calculation shows that

$$H(X \mid Y) = H(X,Y) - H(Y),$$

where $H(X,Y)$ is the entropy of the random variable $Z = (X,Y)$ whose values are pairs (x_i, y_j) of values of X and Y.

We interpret $H(X \mid Y)$ as the remaining uncertainty about X after doing an experiment to measure Y. Indeed, the following holds:

Proposition 3.13 *For any two random variables X and Y, we have $H(X \mid Y) \leq H(X)$, with equality if and only if X and Y are independent.*

Thus, if X and Y are independent, then knowledge of Y gives no information about X.

Let us apply these ideas to cryptography. If we are in Eve's position, we should regard the plaintext, key, and ciphertext as random variables. We will probably have some assumptions about the relative likelihood of various plaintext messages: a spy is unlikely to be sending a passage of Shakespeare as plaintext (though the plaintext may be hidden in passage of Shakespeare, or Shakespeare's works may be used in another way in creating a cipher). This knowledge corresponds to a probability distribution on the plaintexts, from which the entropy $H(P)$ of the plaintext can be calculated. (Here P is the random variable whose values are the actual plaintexts.)

Once Eve intercepts a ciphertext, she can in principle compute some information about the plaintext. This may be complete information (that is, Eve can decrypt the cipher), or perhaps just some change in the probabilities. The conditional entropy $H(P \mid Z)$ is Eve's remaining uncertainty about the plaintext given the ciphertext; it is zero if she can decrypt the message.

In this form, Shannon's theorem states that, if Alice uses a one-time pad, then $H(P \mid Z) = H(P)$: in other words, Eve gets no information about the plaintext from knowledge of the ciphertext.

Exercises

3.1. Prove Proposition 3.12.

3.2. Prove that $H(X \mid Y) = H(X,Y) - H(Y)$.

3.3. Calculate $H(P)$ and $H(P \mid Z)$ in the worked example on page 56.

Chapter 4

Public-key cryptography: basics

In this chapter we describe the revolutionary approach to cryptography that emerged in the second half of the twentieth century: *Public-key cryptography*.

4.1 Key distribution

We have seen that it is possible to construct an 'unbreakable' cipher using randomness: this is the one-time pad, whose key is a string of characters as long as the message.

One weakness of all the ciphers we have studied so far is the problem of *key distribution*. If Eve can get hold of the key, then she can decrypt the cipher. On the other hand, Alice and Bob must both know the key, or they cannot communicate. So they must share the key by some secure method which Eve cannot penetrate.

In the classical field of espionage, a spy is given the key (which might be one copy of the one-time pad, the other copy being held by the home agency) before being sent out into the field. Since the key must not be re-used, the spy can only send as much information as the key he possesses. Then he must return to base for a new one-time pad. This system can work well, if the spy keeps the pad on his person and destroys each page when it is used. One of the stories told by Peter Wright in *Spycatcher* relates how MI6 agents found a one-time pad in the possessions of a suspected spy; they copied the pad and returned it, and were subsequently able to read the communications. Of course, having a one-time pad on your person might be extremely dangerous!

Other ciphers use a key which is smaller than the message. For example, a military commander might be issued with a set of keys, and instructed to use a

new key every month according to some schedule. But if the enemy captures the keys, then all communications can be read until the whole set of keys is changed; this change may be difficult in wartime.

The commercial use of cryptography since the second world war introduced new problems. Commercial organisations need to exchange secure communications; the only way of exchanging keys seemed to be by using trusted couriers. The amount of courier traffic began to grow out of control. It was the invention of public-key cryptography which gave us a way round the key distribution problem.

That there is a possible way around the problem is suggested by the following fable. Alice and Bob wish to communicate by post, but they know that Eve's agents have control of the postal service, and any letter they send will be opened and read unless it is securely fastened. Alice can put a letter in a chest, padlock the chest, and send it to Bob; but Bob will be unable to open the chest unless he already has a copy of Alice's key!

The solution is as follows. Alice puts her letter in the chest, padlocks it and sends it to Bob. Now Bob cannot open the chest. Instead, he puts his own padlock on the chest and sends it back to Alice. Now Alice removes her padlock and returns the chest to Bob, who then simply has to remove his own padlock and open the chest.

A little more formally, let Alice's encryption and decryption functions be e_A and d_A, and let Bob's be e_B and d_B. This means that Alice encrypts the plaintext p as $e_A(p)$; she can also decrypt this to p, which means that $d_A(e_A(p) = p$.

Now Alice wants to send the plaintext p to Bob by the above scheme. She first encrypts it as $e_A(p)$ and sends it to Bob. He encrypts it as $e_B(e_A(p))$ and returns it to Alice. Now we have to make a crucial assumption:

> e_A and e_B commute, that is, $e_A \circ e_B = e_B \circ e_A$.

Now Alice has $(e_B \circ e_A)(p)$, which is equal to $e_A \circ e_B(p) = e_A(e_B(p))$ according to our assumption. Alice can now decrypt this to give $d_A(e_A(e_B(p))) = e_B(p)$ and send this to Bob, who then calculates $d_B(e_B(p)) = p$. At no time during the transaction is any unencrypted message transmitted or any key exchanged.

Note that the operations of putting two padlocks onto a chest do indeed commute! The method would not work if, instead, Bob put the chest inside another chest and locked the outer chest; the operations don't commute in this case.

If the letter that Alice sends to Bob is the key to a cipher (say a one-time pad), then Alice and Bob can now use this cipher in the usual way to communicate safely, without the need for the to-and-fro originally required. The system only

depends on the security of the ciphers used by Alice and Bob for the exchange, and the fact that they commute.

Now if Alice and Bob use binary one-time pads for the key exchange, then these conditions are satisfied, since binary addition is a commutative operation.

However, further thought shows that this is not a solution at all! Suppose that Alice wants to send the string l securely to Bob (perhaps for later use as a one-time pad). She encrypts it as $l \oplus k_A$, where k_A is a random key chosen by Alice and known to nobody else. Bob re-encrypts this as $(l \oplus k_A) \oplus k_B$, where k_B is a random key chosen by Bob and known to nobody else. Now $(l \oplus k_A) \oplus k_B = l \oplus k_B) \oplus k_A$, so when Alice re-encrypts this message with k_A she obtains

$$((l \oplus k_B) \oplus k_A) \oplus k_A = (l \oplus k_B \oplus (k_A \oplus k_A) = l \oplus k_B,$$

and when Bob finally re-encrypts this with k_B he obtains

$$(l \oplus k_B) \oplus k_B = l.$$

This is the exact analogue of the chest with two keys.

If Eve only intercepts one of these three transmissions, it is impossible for her to read the message, since each is securely encrypted with a one-time pad. However, we must assume that Eve will intercept all three transmissions. Now if she simply adds all three together mod 2, she obtains

$$(l \oplus k_A) \oplus (l \oplus k_A \oplus k_B) \oplus (l \oplus k_B) = l,$$

and she has the message!

4.2 Complexity

In trying to wrestle with this problem, Diffie and Hellman came up with an even more radical solution to the problem of key sharing: it is not necessary to share the keys at all! The reason for the insecurity of the above protocol is that decryption is just as simple as encryption for someone who possesses the key; indeed, for binary addition, it is exactly the same operation. (A cipher with this property is called *symmetric*.) The trick is to construct an asymmetric cipher, where decryption is ruinously difficult even if you are in possession of the key.

In order to understand this, we must look at what is meant when we say that a problem is *easy* or *difficult*. This is the subject-matter of *complexity theory*. What

follows is a brief introduction to complexity theory. You can find much more detail either in the lecture notes at
`http://www.maths.qmul.ac.uk/~pjc/notes/compl.pdf`,
or in books such as M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.*

The subject of computational complexity grew out of computability theory, originally due to Alan Turing (who was also one of the most successful cryptanalysts of the twentieth century). Turing succeeded in showing that there are some mathematical problems which cannot be solved by a machine carrying out an algorithm.

In order to demonstrate this, Turing had to analyse the process of computation. He proposed a model, called a *Turing machine*, and showed that it can carry out any process which can be described algorithmically. Said otherwise, a Turing machine can 'emulate' any computer, real or imagined, that has ever been proposed. Seventy years later, despite the efforts of physicists and philosophers, Turing's claim still stands.

A Turing machine consists of two parts: a *tape* and a *head*.

- The tape is made up of cells stretching infinitely far in both directions. Like the RAM or the hard disc of a computer, it stores information; each cell can either be blank or have a symbol from an alphabet A written on it. The one difference between a Turing machine and a real computer is that the tape is infinite; but we assume that only finitely many tape squares are not blank. So we could regard the memory as finite but unbounded; if more memory is needed for a computation, it is always available.

- The head is a machine which can be in any one of a finite number of states; it resembles the CPU of a computer. The head also has access to one square of the tape.

The configuration of the machine is given by describing

- the string of symbols written on the non-blank squares of the tape;

- the state of the head, and its position (the square which it is scanning).

Now the machine operates as follows. It has a program, a finite set of rules determining what it does at any moment. The action is determined by the state of the head and the symbol on the square which it is scanning. The program can

direct the head to change into a specified state, and either to change (or erase) the symbol on the tape square, or to move one place to the left or the right.

One (or more) of the states is distinguished as a 'halting state'. In order to perform a computation, we place a finite amount of information on the tape and put the head in a particular state scanning a particular square. Then the machine starts operating; if it reaches a halting state, its output is the information written on the tape.

Now we can say that a function is computable if there is a Turing machine which computes it. For example, if the tape alphabet is the set of digits $\{0, 1, \ldots, 9\}$, we could design a machine so that, if the number N is written (in the usual way in base 10) on the tape and the machine is started immediately to the right of the string, it calculates N^2, writes the answer on the tape, and halts. All that such a machine needs is an appropriate program (which might, for example, include the usual multiplication table), and it can square a number of any size.

Clearly this is a very basic kind of machine. But adding facilities such as increasing the number of states, or giving it extra tapes (even changing the tape into a two-dimensional array), or allowing the machine to access any tape square within a fixed distance of the head, we do not change the class of computable functions. Turing showed that there exist mathematical functions which are not computable in this sense.

Now complexity changes the question "Can this function be computed?" to the question "How long will it take to compute it?" Variations are possible, such as "How much memory will I need for the computation?" Clearly the precise answers will depend on the precise details of the Turing machine, so we ask the question in a fairly broad-brush way.

First let us be clear that we are not interested in one-off questions of a general kind such as "Is Goldbach's Conjecture true?" A *problem* in this context means a whole class of *problem instances*. We specify a problem by saying what data comprises the problem instance, and what answer we require (which might be just 'Yes' or 'No', or might be some data such as the square of N).

We measure the *size* of a problem instance by the number of tape squares needed to write down the input data. It makes little difference if we decide to use only the binary alphabet, and define the size of a problem instance to be the number of bits of input data. (For example, if we write the number N in base 2 instead of base 10, we need only $\log_2(10) = 2.30\ldots$ times as many tape squares; a constant factor does not matter here.

Now we organise problems into *complexity classes* as in the following examples:

- A problem lies in P, or is *polynomial-time solvable*, if there is a Turing machine which can solve an instance of the problem of size n in at most $p(n)$ for some polynomial p.
- A problem lies in NP, or is *non-deterministic polynomial-time solvable*, if there is a Turing machine which can check the correctness of a proposed solution of a problem instance of size n in at most $p(n)$ steps, for some polynomial p.
- A problem lies in PSpace, or is *polynomial-space solvable*, if there is a Turing machine which can solve an instance of the problem of size n using at most $p(n)$ tape squares, for some polynomial p.
- A problem lies in ExpTime, or is *exponential-time solvable*, if there is a Turing machine which can solve an instance of the problem in at most $2^{p(n)}$ steps, for some polynomial p.

Clearly a problem of higher complexity is harder, and this is a very practical thing to know. If a problem takes n^3 steps to solve, and each step takes a nanosecond, then an instance of size 1000 can be solved in a second, and an instance of size 10000 in three months. However, if it takes 2^n steps, then we can solve an instance of size 30 in a second, while an instance of size 100 will take longer than the age of the universe! The general paradigm is that polynomial-time problems are easy, while exponential-time problems are hard. (Of course much depends on the degree and coefficients of the polynomial; but this works well as a rule of thumb.)

Now we have:

Theorem 4.1 $\mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSpace} \subseteq \mathsf{ExpTime}$.

As this is not a course on complexity, we will not prove this in detail; but a few comments on the proof might help explain the concepts. The first inclusion holds because finding a solution is easier than checking a proposed solution.

The second inclusion holds because, if we can check any proposed solution in polynomial time, the check only use a polynomial number of tape squares. So we simply work through all possible solutions until we find one that works.

The last inclusion follows because, if the alphabet has size q, then $p(n)$ tape squares can only hold at most $q^{p(n)}$ possible strings. If the computation took more than this number of steps, we would have to revisit a previous configuration then the machine would be in an infinite loop, and would not finish at all.

One thing remains to be stressed. It is (relatively) easy to show that a problem lies in a particular complexity class. Strictly, we have to show that there is a Turing machine which solves it efficiently. In practice, it is enough to find some algorithm which solves the question efficiently. Then translating that algorithm into a Turing machine may increase the number of steps, but not enough to affect our broad-brush conclusions.

However, it is very difficult to show that a problem is *not* in a particular complexity class, since we would have to show that no possible Turing machine, or no possible algorithm, can solve the problem efficiently enough. There are many instances of problems where the naive algorithm has been superseded by a much more efficient algorithm.

Thus, it is known that P is properly contained in $\mathsf{ExpTime}$, and so at least one of the inclusions in Theorem 4.1 must be proper. It is conjectured that they are all proper. For example, there are problems in NP (the so-called NP-complete problems) which have been studied for a long time, and nobody has ever managed to find an algorithm to solve any of them in polynomial time.

Our rough equivalences will be:

$$\begin{aligned} \text{'easy'} &= \mathsf{P}, \\ \text{'hard'} &= \mathsf{NP}\text{-complete}. \end{aligned}$$

The NP-complete problems are the 'hardest' problems in NP. If a polynomial-time algorithm were ever found for one of them, then we would conclude that $\mathsf{P} = \mathsf{NP}$. It is conjectured that this is not the case. (The Clay Mathematical Institute has offered one million dollars for a proof or disproof of this, as one of its seven millennial problems.)

4.3 Public-key cryptography

The idea of public-key cryptography based on the fact that there are easy and hard problems was devised by Diffie and Hellman in the 1970s. This is one of the great ideas of the twentieth century!

In order to explain how a cipher can be secure when the key is publicly available, we now formulate the general setup of cryptography a bit more carefully.

Let $\mathcal{P}$ be the set of plaintext messages that users of the system might wish to send. (Thus, $\mathcal{P}$ might be the set of all strings of letters and punctuation marks, or strings of zeros and ones, or certain strings of dots and dashes.) Let $\mathcal{K}$ be the set

of keys, and $\mathcal{Z}$ the set of ciphertexts. Then there is an *encryption function*

$$e : \mathcal{P} \times \mathcal{K} \to \mathcal{Z}$$

and a *decryption function*

$$d : \mathcal{Z} \times \mathcal{K} \to \mathcal{P}$$

which must satisfy the relationship

(PK1) $d(e(p,k),k) = p$.

This simply says that encryption followed by decryption using the same key must recover the original plaintext.

Now the first requirement of public-key cryptography is:

(PK2) Evaluating e should be easy.

(PK3) Evaluating d should be difficult.

(Here, ideally, we should use the equations of the preceding section, that is, 'easy' means 'polynomial-time', while 'hard' means 'NP-complete'. In practice, it almost always means something less precise than this.)

This means that we may assume that Eve not only knows the ciphertext z that Alice sent to Bob, but she also knows the key k and the functions e and d used for encoding and decoding; so all she has to do is to evaluate $d(z,k)$. However, this is a hard problem, and we can assume that, even with the most advanced current technology, it will take her (say) a hundred years to evaluate this function. By that time, the protagonists are all dead and the information has no value.

However, there is a problem here. If decryption is hard, how does Bob (the legitimate recipient) manage to do it? The answer is that there is yet another layer. There is a set $\mathcal{S}$ of *secret keys*, together with an inverse pair of functions

$$g : \mathcal{S} \to \mathcal{K}, \qquad h : \mathcal{K} \to \mathcal{S}.$$

(Think of the mnemonics 'go public' and 'hide'.) Now we make the following requirements:

(PK4) Evaluating the composite function $d^*(z,s) = d(z,g(s))$ is easy.

(PK5) Evaluating g is easy

(PK6) Evaluating h is hard.

Assumption (PK4) means that, given s and z, it is easy to compute p such that $d(z,k) = p$ (or equivalently $e(p,k) = z$) for the unique k which satisfies $h(k) = s$ (or equivalently $g(s) = k$). Note that this does not mean that it is easy to compute $g(s) = k$ and then $d(z,k) = p$, since the latter computation is assumed to be hard; there should be an easy way to compute the composite function d^*.

Now let us see how the system works. Alice wants to send a message to Bob which is secure from the eavesdropper Eve. Bob chooses a 'secret key' from the set $\mathcal{S}$ and tells nobody of his choice. He computes the corresponding 'public key' $k = g(s) \in \mathcal{K}$ and makes this available to Alice. Bob is aware that Eve will also have access to his public key k. We observe that this computation is assumed to be easy.

Alice wants to send Bob the plaintext message p. Knowing his public key k, she computes the ciphertext $e(p,k)$ and sends this to Bob. (This computation is also easy.)

Bob is now faced with the problem of decrypting the message. But Bob already knows the secret key s, and so he only has to do the easy computation of $p = d^*(z,s)$. Since $g(s) = k$, we have $p = d(z,k)$, so that p is indeed the correct plaintext that Alice wanted to send.

What about Eve? Her position is different, since she doesn't know the secret key. Either she has to compute $d(z,k)$ directly (which is hard), or she could decide to compute Bob's secret key s by evaluating the function $s = h(k)$ (which is also hard).

Note that Eve knows in principle how to evaluate either of these functions; the only thing keeping the cipher secure is the complexity of the computations. The important thing is that the secret key, which enables Bob to decrypt the message, is never communicated to anyone else; Bob chooses it, and uses it only to decrypt messages sent to him.

Now in principle we have a method for any set of people to communicate securely. Suppose we have a number of users $A, B, C, \ldots$. Each user chooses his or her own secret key: thus, Alice chooses s_A, Bob chooses s_B, and so on. These choices are never communicated to anyone else. Now Alice computes $k_A = g(s_A)$ and publishes it; and similarly Bob computes $k_B = g(s_B)$ and so on. Then anyone who wishes to send a message p to Alice first obtains her public key k_A (which may be in a directory or on her Web page), and then encrypts it as $z = e(p,k_A)$ and transmits this to Alice. She can calculate $p = d^*(z,s_A) = d(z,k_A)$; but nobody else can read the message without performing a hard calculation.

Some terminology that is often used here is that of 'one-way functions'. A function $f : A \to B$ is said to be *one-way* if it is easy to compute f but hard to

compute the inverse function from B to A. It is a *trapdoor one-way function* if there is a piece of information which makes the computation of the inverse function easy. Thus, for public-key cryptography, we want encryption to be a trapdoor one-way function, where the key to the trapdoor is the secret key; the function from secret key to public key should be a one-way function.

4.4 Digital signatures

There is a serious potential weakness of public-key cryptography. Eve cannot read Alice's message to Bob. But, since Bob's key is public, Eve can write her own message to Bob purporting to come from Alice, encrypt it with Bob's key, and substitute it for Alice's authentic message on the communication channel. Is there a way around this?

Indeed there is. We make two further assumptions, namely:

(PK7) The set $\mathcal{P}$ of plaintext messages is the same as the set $\mathcal{Z}$ of ciphertexts.

(PK8) $e(d(z,k),k) = z$ for any $z \in \mathcal{Z}$ and $k \in \mathcal{K}$.

The first assumption is not at all restrictive. Almost always, in practice, both sets will consist of all binary strings. The second assumption strictly follows from the others. Condition (PK1) says that decryption is the inverse of encryption; that is, the functions $p \mapsto e(p,k)$ and $z \mapsto d(z,k)$ are inverse bijections (the second undoes the effect of the first). Now inverse functions on finite sets work 'both ways round', so the first undoes the effect of the second; this is exactly what (PK8) claims. The reason that we make this assumption is that in practice the functions may not quite be bijections, or the sets of potential plaintexts may be infinite.

Alice wants to send the plaintext p to Bob, in such a way that it cannot be faked by Eve. First, bizarrely, she pretends that p is a ciphertext and *decrypts it using her own secret key*! In other words, she computes $u = d(p,k_A)$. The result, of course, appears to be gibberish.

Now she writes a preamble in plaintext saying "This is a signed message from Alice", and now encrypts the whole thing using Bob's public key; that is, she calculates $z = e(u,k_B) = e(d(p,k_A),k_B)$. She sends this message to Bob.

Now Bob decrypts this message using his own secret key, obtaining $d(z,k_B) = u$. He sees the statement "This is a signed message from Alice", followed by some gibberish. Now he does another strange thing: he *encrypts* the gibberish, using

Alice's public key (as if it were a message he wanted to send to Alice). This gives $e(u,k_A)$, which is equal to p by our assumption (PK8) (since $d(p,k_A)=u$). Then Bob has the intended message.

Assumption (PK8) further tells us that the equation $e(u,k_A)=p$ is equivalent to $d(p,k_A)=u$. Thus, the only person who could compute this is the holder of Alice's secret key, namely Alice herself; so Bob is assured that the message is from Alice. (For Eve to fake such a message, she is faced with the same problem as in decrypting a message from Alice, that is, either compute $d(p,k_A)$, or compute $h(k_A)$; both are hard problems.)

4.5 The knapsack cipher

We have seen how a scheme for public-key cryptography can be designed, based on trapdoor one-way functions (and so ultimately on the existence of functions which are hard to compute). Now we turn to the question of finding practical examples on which to base a cipher system. In this chapter we look at a couple of examples which have not caught on; in the next, we turn to two of the most popular schemes, RSA and El-Gamal.

One of the earliest problems to be shown to be NP-complete was the *knapsack problem*. Unofficially, we are given a knapsack with a volume of b units, and items of volume $a_1,a_2,\ldots,a_k$ units. We want to know whether we can fill the knapsack using some of the items.

More formally, the input data for this problem consists of the number b and the list $(a_1,a_2,\ldots,a_k)$ of numbers. Since a number between 2^m and $2^{m+1}-1$ can be written in base 2 using m bits, we see that the size of a number a when regarded as input data is about $\log_2(a)$, and so the size of the data for this problem is about

$$\log_2(b)+\sum_{i=1}^{k}\log_2(a_i).$$

We are asked to find a k-tuple $(e_1,e_2,\ldots,e_k)$, where each e_i is equal to 0 or 1, such that

$$\sum_{i=1}^{k}e_ia_i=b$$

if possible, or discover that no such tuple exists. This problem is in NP, since we can very easily check a purported solution by simple arithmetic. But finding a

solution is harder. In principle, we have 2^k possible k-tuples to check, and if there is no solution we might have to look at all of them. This is not a proof that the problem is hard, since there may be a smarter way to do it; but this problem is indeed known to be hard:

Theorem 4.2 *The knapsack problem is* NP*-complete.*

Recall that this theorem makes two assertions:

(a) The problem is in NP; that is, we can *check* whether a proposed solution $(e_1, e_2, \ldots, e_k)$ is correct in a polynomial number of steps. (The check is just integer addition!)

(b) If an algorithm to *solve* the problem in a polynomial number of steps were found, then we would know that P = NP (which is believed not to be the case).

For example, suppose that we are given the list

$$(323, 412, 33, 389, 544, 297, 360, 486)$$

and a target number 1228. If we try the *greedy algorithm*, which says "at each stage, put the largest item which will fit into the knapsack", we obtain

$$1228 = 544 + 684 = 544 + 486 + 198 = 544 + 486 + 33 + 165,$$

and then we are stuck. So the greedy algorithm fails to solve the problem.

In the end, exhaustive search of some kind reveals that

$$1228 = 412 + 33 + 297 + 486.$$

As can be imagined, a similar problem with 100 numbers of 50 digits each would present quite formidable difficulty.

Now we can make a cipher based on this hard problem as follows. The public key consists of a k-tuple $(a_1, a_2, \ldots, a_k)$ of integers. In order to encrypt a message, we first write it as a string of bits, and break it into blocks of length k. Now the block $(e_1, e_2, \ldots, e_k)$ is encrypted as the integer

$$a = \sum_{i=1}^{k} e_i a_i = b,$$

and this integer is transmitted.

In order to break the cipher it is necessary to solve this instance of the knapsack problem, which is hard! Of course, we also need a secret key so that the intended recipient can decrypt the message.

The way the key is constructed illustrates one important thing about computational complexity, which we haven't stressed so far. For a problem to be easy, it is necessary that there is an algorithm which solves *any* instance efficiently. It may be that some (perhaps just a few) instances are hard; then the problem will be classified as hard, even if most cases are actually easy. In other words, we are measuring 'worst-case complexity' rather than 'average-case complexity'.

Now there are indeed some instances of the knapsack problem which are easy to solve. These correspond to the so-called super-increasing sequences.

The sequence $(a_1, a_2, \ldots, a_k)$ of positive integers is called *super-increasing* if each term is greater than the sum of its predecessors, that is, if

$$\sum_{j=1}^{i-1} a_j < a_i$$

for $i = 1, \ldots, k$. If the data in the knapsack problem is super-increasing, then the greedy algorithm we met earlier, that is, "put into the knapsack the largest object which will fit", is guaranteed to solve the problem. In other words, let i be the largest index for which $a_i \leq b$; then set $e_i = 1$ and $e_j = 0$ for $j > i$, and (recursively) solve the knapsack problem for the integer $b - a_i$ with the sequence $(a_1, \ldots, a_{i-1})$. The reason for this is that, if the ith item is the largest one which fits in the knapsack, then we must use it; the larger objects don't fit and, even if all the smaller objects were used, they would not fill the knapsack. (This argument shows a bit more: if a solution exists, then it is unique.)

For example, the sequence $1, 2, 4, 8, \ldots$ of powers of 2 is super-increasing; the above algorithm is exactly what we do when we express an integer in base 2. For example,

$$27 = 16 + 11 = 16 + 8 + 3 = 16 + 8 + 2 + 1,$$

where we take at each step the largest power of 2 not exceeding what we have left.

We cannot just use a super-increasing sequence as public key, since Eve could recognise that it is super-increasing and use the greedy algorithm to decrypt the cipher. So we have to disguise it. This can be done as follows. Bob chooses a super-increasing sequence $(a_1, a_2, \ldots, a_k)$. Then he chooses an integer $n > \sum a_i$ and an integer u with $\gcd(n, u) = 1$, and builds the new sequence $(a_1^*, a_2^*, \ldots, a_k^*)$,

where

$$a_i^* = ua_i \bmod n$$

for $i = 1, \ldots, k$. It is very unlikely that these numbers will still be super-increasing, so Bob can use them as the public key.

Now to encipher the binary string $(e_1, \ldots, e_k)$, Alice computes $b^* = \sum e_i a_i^*$, and sends this to Bob. To decrypt this, he calculates the inverse v of u mod n, using Euclid's Algorithm (as we have seen before). Then he calculates $b = vb^* \bmod n$. Now we have

$$\begin{aligned} b &\equiv vb^* \pmod{n} \\ &= v\sum e_i a_i^* \\ &\equiv v\sum e_i(ua_i) \pmod{n} \\ &= (uv)\sum e_i a_i \\ &\equiv \sum e_i a_i \pmod{n}. \end{aligned}$$

But both b and $\sum e_i a_i$ are smaller than n. (Remember that we chose $n > \sum a_i$.) So, if they are congruent mod n, then they are actually equal:

$$b = \sum e_i a_i.$$

So Bob has only to solve an easy instance of the knapsack problem (with super-increasing data) in order to decrypt the message.

For example, suppose that we take the super-increasing sequence

$$(1, 3, 7, 15, 31, 63, 127, 255).$$

Take the modulus 557, which is greater than the sum of the terms in the sequence, and multiply by the coprime inteteger 323 to get the sequence

$$(323, 412, 33, 389, 544, 297, 360, 486).$$

Now the bit string 01100101 (character e in 8-bit ASCII) is encoded as $412 + 33 + 297 + 486 = 1228$. To decrypt this without solving a 'hard' instance of the knapsack problem, Bob knows that the inverse of 323 mod 557 is 169 (having found that $169 \cdot 323 - 98 \cdot 557 = 1$); then he calculates $1228 \cdot 169 \bmod 557$, which is 328; and then he applies the greedy algorithm to get

$$328 = 255 + 73 = 255 + 63 + 10 = 255 + 63 + 7 + 3$$

so that the bit string is 01100101 as required.

For added security one can apply the 'disguising' transformation of multiplying by u mod n several times over (with different choices of n and u) before publishing the key.

This was the first practical public-key cryptosystem to be proposed; it was invented by Merkle and Hellman, soon after the basic principles of public-key cryptography had been stated by Diffie and Hellman. It is not actually used today. The problem is it is thought that keys obtained by disguising super-increasing sequences in this way are somehow special, and the knapsack problem for such keys may turn out to be easier than it is for completely general instances of the knapsack problem. Once any doubt has been cast on a cipher system, people are reluctant to use it!

4.6 A cipher using a code

Another system was proposed by McEleice, based on the theory of error-correcting codes. This is an entirely different topic, which we summarise briefly. Consider the following list of sixteen binary strings of length 7:

$$
\begin{array}{ccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 1 & 1 \\
0 & 1 & 0 & 1 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 1 & 1 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 \\
0 & 1 & 1 & 0 & 0 & 1 & 1 \\
0 & 1 & 1 & 1 & 1 & 0 & 0 \\
1 & 0 & 1 & 0 & 1 & 0 & 1 \\
1 & 0 & 1 & 1 & 0 & 1 & 0 \\
1 & 1 & 0 & 0 & 1 & 1 & 0 \\
1 & 1 & 0 & 1 & 0 & 0 & 1
\end{array}
$$

A little checking shows that any two of these 7-tuples differ in at least three positions. This means that, if one of them is transmitted through a noisy channel

which might make a single *error* (that is, change a 0 to a 1 or *vice versa*), the received sequence will still be closer to the transmitted sequence than to any other sequence in the list.

The sixteen 7-tuples have another important property. They consist of all possible linear combinations of four of them (over the integers mod 2); that is, they form a 4-dimensional subspace of the 7-dimensional vector space over $\mathrm{GF}(2)$, the field of integers mod 2. We can take a basis as the rows of the matrix

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

This provides a very simple way to encode information. If the message to be transmitted is the binary 4-tuple $e = (e_1, e_2, e_3, e_4)$, then we encode it as the 7-tuple

$$eG = e_1a_1 + e_2a_2 + e_3a_3 + e_4a_4$$

using matrix multiplication over $\mathrm{GF}(2)$ (where $a_1, \ldots, a_4$ are the rows of G).

Decoding is more difficult, since (assuming that an error might have occurred) we have in principle to compare the received word to all 16 codewords to see which is nearest.

We can generalise all this. If G is a $k \times n$ matrix over $\mathrm{GF}(2)$ with rank k, then we can encode a binary k-tuple e into an n-tuple eG by matrix multiplication, which is easy. If some errors occur (in a pattern which the code can correct), then to decode we must find the particular one of the 2^k codewords which is nearest to the received word. This looks hard; and indeed it has been shown that the problem of decoding an arbitrary linear code is NP-complete.

However, there some codes with particular algebraic structure for which efficient decoding algorithms exist. These are widely used in practice; for example, Reed–Solomon codes in CD players, Reed–Muller and Golay codes in space probes.

Our small example gives us an indication of how there can be a 'hard way' and an unexpected 'easy way' to decode. Suppose we are using the 16-word code of length 7 given earlier. The hard way to decode is to compare the received word with each transmitted word to find out which is nearest. For example, if (0111001) is received then we find that the seventh row of the table, (0011001), differs from it in the second position, and must be the transmitted word (assuming

at most one error). However, the following *syndrome decoding* method is more straightforward. Let H be the matrix

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

If the received word is v, we calculate vH, which is a string of three bits. Regard this string as the base 2 representation of an integer m in the range $0 \ldots 7$. If $m = 0$, then the received word is correct; if $m = 1$, there is an error in the mth position.

In our case,

$$(0,1,1,1,0,0,1)H = (0,1,0)$$

and $(0,1,0)$ is the number 2 in base 2, so the second digit is wrong.

You might like to try to explain why this works. This material is covered in the Coding Theory course (MAS309), or in books such as Ray Hill, *A First Course in Coding Theory*.

McEleice's idea is to use the fact that encoding is easy and decoding is difficult as the base of a public-key cipher.

Suppose that Alice wants to send a message to Bob. First, Bob chooses a large code for which an efficient decoding algorithm exists. He also chooses a random permutation and applies it to the columns of the matrix G. The resulting matrix G^* is the public key.

If Alice wants to send the binary k-tuple e to Bob, she first calculates eG^*, and then randomly changes a few of the entries (this corresponds to making some random errors). This is transmitted to Bob.

By applying the inverse of his permutation to the cipher, Bob obtains a word encoded using G, which he can decode efficiently (correcting the errors at the same time!) using the decoding algorithm for G.

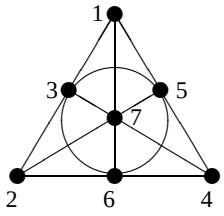
However, Eve is faced with decoding a word encoded with G^*, which looks like an 'arbitrary' linear code. Without the benefit of the algebraic structure, it is hard to decode.

In terms of the last section of the notes, the encryption function is just matrix multiplication $e \mapsto eG$. Decryption consists of error-correction followed by recovering e from eG. The function g from secret key to public key is applying a

permutation to the columns of G; the inverse function involves finding a permutation which converts the 'unknown' code into one for which an efficient decoding algorithm is known.

Any public-key cipher can be attacked in two ways: either try to decrypt directly, or try to reconstruct the private key from the public key. In the case of McEleice's cipher, the latter attack is more likely. We may be able to use the structure of the code in some way.

In our example, some sets of four columns are linearly independent and some are linearly dependent. If we take the set of triples of columns whose complements are linearly dependent, we get a recognisable picture which gives the structure of the code:



Even if the code is presented in arbitrary order, we can build a similar picture and map it onto this one; this will tell us how to rearrange the columns into an order for which our syndrome decoding algorithm will work.

Exercises

4.1. I claim that

$$37332305417280604729 = 7392847577 \times 5049786977$$

and that the two factors are prime. About how many arithmetic operations are required

1. to check that my multiplication is correct,

2. to check that the factors are prime,

3. to find the factorisation in the first place?

(You may take an arithmetic operation to be a single addition, subtraction, multiplication or division of integers.)

4.2. For Alice and Bob to share a key over an insecure channel, it is necessary that their encryption functions should commute with each other. In which of the following cases does this condition hold?

1. Caesar shifts;
2. affine substitutions;
3. arbitrary substitutions;
4. stream ciphers with alphabet $\{0,\ldots,q-1\}$, where the substitution table is addition mod q;
5. stream ciphers with arbitrary alphabet and arbitrary substitution table.

Which of these would be suitable for actual use?

Chapter 5

Public-key cryptography: RSA and El-Gamal

In this chapter we will describe the RSA and El-Gamal cryptosystems, the most popular public-key cryptosystem at present. We need some number-theoretic background first.

5.1 More number theory

The RSA enciphering function has the form $T_d : x \mapsto x^d \bmod n$ for some suitable n and d. In order to be able to decipher, we must be assured that this function is one-to-one. So we first discuss the number-theory required for this question.

Euler and Carmichael

Recall Euler's phi-function $\phi(n)$ whose value is the number of elements of $\mathbb{Z}/(n)$ (the integers mod n) which are coprime to n. We calculated this function back in Notes 2:

Theorem 5.1 *(a) If $n = p_1^{a_1} \cdots p_r^{a_r}$, where p_i are distinct primes and $a_i > 0$, then $\phi(n) = \phi(p_1^{a_1} \cdots \phi(p_r^{a_r})$.*

(b) If p is prime and $a > 0$, then $\phi(p^a) = p^a - p^{a-1} = p^{a-1}(p-1)$.

A well-known theorem of Fermat (often called *Fermat's Little Theorem*) asserts that, if p is prime, then $a^{p-1} \equiv 1 \pmod{p}$ for any number a not divisible by p. This theorem was generalised by Euler as follows:

Theorem 5.2 *If* $\gcd(a,n) = 1$, *then* $a^{\phi(n)} \equiv 1 \pmod{n}$.

Proof Let $x_1, x_2, \ldots, x_m$ be all the elements of $\mathbb{Z}/(n)$ which are coprime to n, where $m = \phi(n)$. Suppose that $\gcd(a,n) = 1$. Then a has an inverse b mod n, so that $ab \equiv 1 \pmod{n}$. Now let $y_i = ax_i \bmod n$, and consider $y_1, \ldots, y_m$. We have

- $\gcd(y_i, n) = 1$, since $\gcd(a,n) = \gcd)x_i, n) = 1$;
- $y_1, \ldots, y_m$ are all distinct: for if $y_i = y_j$, then $by_i = by_j$, that is, $bax_i \equiv bax_j \pmod{n}$, or $x_i \equiv x_j \pmod{n}$, so $x_i = x_j$.

Thus, the set $\{y_1, \ldots, y_m\}$ is the same as the set $\{x_1, \ldots, x_m\}$ (possibly in a different order), so their products are the same:

$$\prod x_i = \prod y_i \equiv \prod ax_i = a^m \prod x_i \pmod{n}.$$

Since the x_i are coprime to n, so is their product, and it has an inverse mod n. Multiplying the equation by this inverse we get $a^m \equiv 1 \pmod{n}$, as required.

One very important fact about Fermat's Little Theorem is that it cannot be improved:

Theorem 5.3 *Let* p *be prime. Then there exists* a *such that* $a^{p-1} \equiv 1 \pmod{p}$ *but no smaller power of* a *is congruent to* 1 *mod* p.

Such an element a is called a *primitive root* or *primitive element* mod p. Since all its powers up to the $p-2$nd are distinct, we see that every non-zero element of $\mathbb{Z}/(p)$ can be expressed as a power of a. This is very similar to a theorem we stated without proof for finite fields in Notes 5; the proof given here easily adapts to the result for finite fields as well. (Note that the integers modulo a prime do form a field; this is used in the proof.)

For example, the powers of 3 mod 7 are

$$3^1 \equiv 3, \quad 3^2 \equiv 2, \quad 3^3 \equiv 6, \quad 3^4 \equiv 4, \quad 3^5 \equiv 5, \quad 3^6 \equiv 1 \pmod{7}$$

so that 3 is a primitive root of 7. But 2 is not a primitive root of 7, since $2^3 \equiv 1$.

Proof We begin with a couple of examples to get the feel of the problem. Suppose that $p = 17$. Then the order of every non-zero element mod p divides 16. If there is no primitive element (one of order exactly 16), then the order of every element would divide 8. But the polynomial $x^8 - 1$ has at most 8 roots in the field $\mathbb{Z}/(17)$, so this can't be the case.

Next consider $p = 37$. The orders of all elements must divide 36; if the order of an element is smaller than 36, then it must divide either 12 or 18. But there are at most $12 + 18$ such elements (arguing as above), so there must be a primitive element.

In general we need to refine this crude counting a bit. Here is the general proof.

Let a be any element with $\gcd(a, p) = 1$. We define the *order* of a mod p to be the smallest positive integer m such that $a^m \equiv 1 \pmod{p}$. In the proof below, we write equality in place of congruence mod p for brevity, so that this condition will be written $a^m = 1$.

Now the order of any element divides $p-1$. For suppose that a has order m, where $p - 1 = mq + r$ and $0 < r < m$. Then

$$1 = a^{p-1} = (a^m)^q \cdot a^r = a^r,$$

contradicting the definition of m. So $r = 0$ and m divides $p - 1$.

Given a divisor m of $p-1$, how many elements of order m are there? Let $f(m)$ be this number. Now we have:

- $f(m) \leq \phi(m)$ for all m dividing $p - 1$. For this is clearly true if $f(m) = 0$, so suppose not. Then there exists some element a with order m. Now the elements $a^0 = 1, a^1, \ldots, a^{m-1}$ are all distinct and satisfy the polynomial equation $x^m - 1 = 0$. But a polynomial of degree m over a field has at most m roots; so these are all the roots. Now it is easy to see that a^r has order m if and only if $\gcd(r, m) = 1$; so there are exactly $\phi(m)$ elements of order m in this case.

- $\sum_{m|p-1} f(m) = p - 1$. This is because each of the $p - 1$ non-zero elements of $\mathbb{Z}/(p)$ has some order!

- $\sum_{m|p-1} \phi(m) = p - 1$. This follows from the fact that the number of integers a with $0 \leq a \leq p - 1$ and $\gcd(a, p - 1) = (p - 1)/m$ is precisely $\phi(m)$, which

is quite easy to see; check it for yourself using the fact that $\gcd(a, p-1) = (p-1)/m$ if and only if $\gcd(a/m, (p-1)/m) = 1$.

From these three statements it follows that $f(m) = \phi(m)$ for all m dividing $p-1$. In particular, $f(p-1) = \phi(p-1) > 0$, so there are some elements which have order $p-1$, as required.

Our proof actually shows us a little more: the number of primitive roots of the prime p is $\phi(p-1)$. For example, $\phi(7-1) = \phi(2 \cdot 3) = 2$, so there are two primitive roots of 7, namely 3 and 5.

Now it is not true that Euler's extension of the little Fermat theorem is best possible. For example, suppose that $\gcd(a, 35) = 1$. Then $\gcd(a, 7) = 1$, so $a^6 \equiv 1 \pmod 7$. Similarly, $\gcd(a, 5) = 1$, so $a^4 \equiv 1 \pmod 5$. From this we deduce that $a^{12} \equiv 1 \pmod 7$ and $a^{12} \equiv 1 \pmod 5$, so $a^{12} \equiv 1 \pmod{35}$. On the other hand, $\phi(35) = \phi(7) \cdot \phi(5) = 6 \cdot 4 = 24$, so Euler only guarantees that $a^{24} \equiv 1 \pmod{35}$.

Carmichael's lambda-function $\lambda(n)$ is defined to be the least number m such that $a^m \equiv 1 \pmod n$ for all a such that $\gcd(a, n) = 1$. It follows from Euler and the argument we used above that $\lambda(n)$ always divides $\phi(n)$, but it may be strictly smaller; for example, $\phi(35) = 24$ but $\lambda(35) = 12$. (We can see that $\lambda(35)$ cannot be less than 12 since, for example, $2^6 \equiv 29 \pmod{35}$ and $2^4 \equiv 16 \pmod{35}$.)

Theorem 5.4 *(a) If $n = p_1^{a_1} \cdots p_r^{a_r}$, where p_i are distinct primes and $a_i > 0$, then $\lambda(n) = \operatorname{lcm}\{\lambda(p_1^{a_1}, \ldots, \lambda(p_r^{a_r})\}$.*

(b) If p is an odd prime and $a > 0$, then $\lambda(p^a) = \phi(p^a) = p^a - p^{a-1} = p^{a-1}(p-1)$.

(c) $\lambda(2) = 1$, $\lambda(4) = 2$, and $\lambda(2^a) = 2^{a-2}$ for $a \geq 3$.

The fact that $\lambda(p) = p-1$ for all primes p is a consequence of Theorem 5.3. Fermat tells us that $a^{p-1} \equiv 1 \pmod p$ for all a coprime to p, and the theorem tells us that no smaller exponent will do.

Suppose that n is the product of two distinct primes, say $n = pq$. The theorem asserts that $\lambda(n) = \operatorname{lcm}(p-1, q-1)$. To show this, let $m = \operatorname{lcm}(p-1, q-1)$. Now, for any integer x coprime to n, we have $x^{p-1} \equiv 1 \pmod p$, and so $x^m \equiv 1 \pmod p$, since $p-1$ divides m. Similarly $x^m \equiv 1 \pmod q$. By the Chinese Remainder Theorem, $x^m \equiv 1 \pmod n$.

In the converse direction, suppose that a is primitive element mod p, and b a primitive element mod q. Use the Chinese Remainder Theorem to find c such that

$$c \equiv a \pmod{p}, \qquad c \equiv b \pmod{q}.$$

Then it is easy to see that the order of c mod n is a multiple both of $p-1$ and of $q-1$, and hence of m. So m is the smallest positive number such that $x^m \equiv 1 \pmod{n}$ for all x coprime to n; that is, $\lambda(n) = m$.

We will not need the other cases of the above theorem.

For example, we have $\lambda(35) = \mathrm{lcm}(\lambda(7), \lambda(5)) = \mathrm{lcm}(6,4) = 12$, as we found earlier.

Power maps

Now consider the transformation

$$T_d : x \mapsto x^d \bmod n.$$

First, we shall simply consider this transformation acting on the set

$$U(n) = \{x \in \mathbb{Z}/(n) : \gcd(x,n) = 1$$

of x with $\gcd(x,n) = 1$. (Note that if $\gcd(x,n) = 1$ then $\gcd(x^d,n) = 1$ for all d.)

Proposition 5.5 *The transformation T_d is one-to-one on $U(n)$ if and only if d satisfies* $\gcd(d,\lambda(n)) = 1$. *So the number of d for which T_d is one-to-one is $\phi(\lambda(n))$.*

We will prove this just in the reverse direction. Suppose that $\gcd(d,\lambda(n)) = 1$. Then there exists e with $de \equiv 1 \pmod{\lambda(n)}$. Then, since $x^{\lambda(n)} = 1$, we have $x^{de} = x$ for all x coprime to n; that is, $T_e T_d$ is the identity map, and so T_d has an inverse.

For example, for $n = 35$, the invertible maps are T_1, T_5, T_7 and T_{11}. The map T_{13} is equal to T_1 on $U(35)$ since $x^{12} \equiv 1 \pmod{35}$ for any $x \in U(35)$.

This condition is not sufficient for T_d to be one-to-one on the whole of $\mathbb{Z}/(n)$. For example, take $n = 9$. Then $\lambda(n) = \phi(n) = 6$, and the number $d = 5$ saatisfies $\gcd(d,\lambda(n)) = 1$. Now the fifth powers mod 9 are given in the following table:

x	0	1	2	3	4	5	6	7	8
$x^5 \bmod 9$	0	1	5	0	7	2	0	4	8

So, in accordance with Proposition 5.5, T_5 is one-to-one on $\{1,2,4,5,7,8\}$ (the numbers coprime to 9); but it maps all the others to zero.

However, there is a special case where we can guarantee that T_d is invertible on $\mathbb{Z}/(n)$:

Proposition 5.6 *Let n be the product of distinct primes. If $\gcd(d,\lambda(n)) = 1$, then $T_d : x \mapsto x^d \bmod n$ is one-to-one on $\mathbb{Z}/(n)$.*

Here is the proof in the case that n is the product of two primes. (This is the only case that is required for RSA, but the proof can be modified to work in general.)

We use the fact that $x^p \equiv x \pmod{p}$ for any prime p. (If p doesn't divide x, this follows from Fermat's little theorem; if $p \mid x$ it is trivial.) Hence $x^{k(p-1)+1} \equiv x \pmod{p}$ for any $k > 0$.

Now, if e is the inverse of $d \bmod \lambda(n)$, then $de \equiv 1 \pmod{\lambda(n)}$, and hence $de \equiv 1 \pmod{p-1}$, since $p-1$ divides $\lambda(n)$. From the preceding paragraph, we see that $x^{de} \equiv x \pmod{p}$. Similarly $x^{de} \equiv 1 \pmod{q}$, and so $x^{de} \equiv 1 \pmod{n}$, by the Chinese Remainder Theorem.

For example, suppose that $n = 15$. Then $\lambda(n) = \operatorname{lcm}(2,4) = 4$, and we can choose $d = 3$. The table of cubes mod 15 is:

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$x^3 \bmod 15$	0	1	8	12	4	5	6	13	2	9	10	11	3	7	14

We see that T_3 is indeed one-to-one.

5.2 The RSA cryptosystem

Preliminaries

The system depends on the following problems. The easy problems are all in P. Unfortunately the hard problems are not known to be NP-complete!

Easy problems

(1) Test whether an integer N is prime.

(2) Given a and n, find $\gcd(a,n)$ and (if it is 1) find an inverse of $a \bmod n$.

(3) Calculate the transformation $T_d : x \mapsto x^d \bmod N$.

Hard problems

(4) Given an integer N, factorise it into its prime factors.

(5) Given an integer N, calculate $\lambda(N)$ (or $\phi(N)$).

(6) Given N and d, find e such that T_e is the inverse of T_d mod N.

Notes about the easy problems My job is to persuade you that they are easy, not to give formal proofs that they belong to the class P.

Problem (1): Note that trial division does not solve this problem efficiently. For a number N requiring n bits of input is one which has n digits when written in base 2, and hence is of size roughly 2^n; its square root is about $2^{n/2}$, and trial division would require about half this many steps in the worst case. Only in 2002 was an algorithm found which solves this problem in a polynomial number of steps, by Manindra Agrawal, Neeraj Kayal and Nitin Saxena at the Indian Institute of Technology, Kanpur. However, the result had been widely expected, since 'probabilistic' algorithms which tested primality with an arbitrarily small chance of giving an incorrect answer have been known for some time. We will consider the question of primality testing further at the end of this chapter.

Problem (2): This is solved by Euclid's algorithm, as we have seen.

Problem (3). On the face of it, this problem seems hard, for two reasons:

- First, the number x^d will be absolutely vast, with about $d \log x$ digits (and remember that the number of digits of d is part of the size of the input; if d has 100 digits, then x^d has too many digits to write down even if the whole universe were our scrap paper).

- Second, we have on the face of it to perform $d-1$ multiplications to find

$$x^d = x \cdot x \cdot x \cdots x \qquad d \text{ factors.}$$

But these difficulties can both be overcome:

Proposition 5.7 *The number a^d mod n can be computed with at most $2\log_2 d$ multiplications of numbers smaller than n and the same number of divisions by n; this can be done in a polynomial number of steps.*

The first difficulty is easily dealt with: we do all our calculations mod n. Thus, to calculate ab mod n, where $a, b < n$, we calculate ab as an integer, and take the remainder on division by n. We never have to deal with a number larger than n^2 in the calculation.

We can reduce this number of multiplications required from $d-1$ to at most $2\log_2 d$ as follows.

Write d in base 2: $d = 2^{a_1} + 2^{a_2} + \cdots + 2^{a_k}$. Suppose that a_1 is the greatest exponent. Then $k \leq a_1 + 1$ and $a_1 \leq \log_2 d$.

By $a_1 - 1$ successive squarings, calculate $x^2, x^{2^2}, \ldots, x^{2^{a_1}}$.

Now

$$x^d = x^{2^{a_1}} \cdot x^{2^{a_2}} \cdots x^{2^{a_k}}$$

can be obtained by $k-1$ further multiplications. The total number of multiplications required is $a_1 + k - 2 < 2\log_2 d$.

This informal description of the algorithm can be translated into a formal proof that problem (3) can be solved in polynomial time.

For example, let us compute $123^{321} \pmod{557}$.

First we find by successive squaring

i	0	1	2	3	4	5	6	7	8
$123^{2^i} \bmod 557$	123	90	302	413	127	533	19	361	540

Now $321 = 2^8 + 2^6 + 1$, so two further multiplications mod 557 give

$$123^{321} \equiv 540 \cdot 19 \cdot 123 \equiv 234 \cdot 123 \equiv 375 \pmod{557}.$$

Notes about the hard problems Problems (4)–(6) are not known to be NP-complete, so it is possible that they may not be as hard as we would like. However, centuries of work by mathematicians has failed to discover any 'easy' algorithm to factorise large numbers. (We will see later that the advent of quantum computation would change this assertion!)

We will be concerned only with numbers N which are the product of two distinct primes p and q. So we really need the special case of (4) which asks:

> Given a number N which is known to be the product of two distinct prime factors, find the factors.

Even this problem is intractable at present.

However, if we know that N is the product of two distinct primes, then problems (4) and (5) are equivalent, in the sense that knowledge of a solution to one enables us to solve the other.

Proposition 5.8 *Suppose that N is the product of two distinct primes. Then, from any one of the following pieces of information, we can compute the others in a polynomial number of steps:*

- *the prime factors of N;*
- *$\phi(N)$;*
- *$\lambda(N)$.*

For suppose first that $N = pq$ where p and q are primes (which we know). Then $\phi(N) = (p-1)(q-1)$ can be found by simple arithmetic. Also, $\lambda(N) = \mathrm{lcm}(p-1, q-1) = (p-1)(q-1)/\gcd(p-1, q-1)$; the greatest common divisor can be found by Euclid's Algorithm, and the rest is arithmetic.

Suppose that we know $\phi(N)$. Then we know the sum and product of p and q, (namely, $p+q = N - \phi(N) + 1$ and $pq = N$); and so the two factors are roots of the quadratic equation

$$x^2 - (N - \phi(N) + 1)x + N = 0,$$

which can be solved by arithmetic (using the standard algorithm for finding the square root).

The case where we know N and $\lambda(N)$ is a bit more complicated. Suppose that p is the larger prime factor. Then $\lambda(N) = \mathrm{lcm}(p-1, q-1)$ is a multiple of $p-1$, and divides $\phi(N)$. Let $r = N \bmod \lambda(N)$ be the remainder on dividing N by $\lambda(N)$. Then

- $N - \phi(N) \equiv r \pmod{\lambda(N)}$, since $\lambda(N) \mid \phi(N)$;
- $N - \phi(N) = p + q - 1 < 2\lambda(N)$, since $\lambda(N) \geq p - 1 > q$ (assuming that $N > 6$).

So $N - \phi(N) = r$ or $N - \phi(N) = r + \lambda(N)$. We can solve the quadratic for each of these two possible values of $\phi(N)$; one of them will give us the factors of N.

Example Suppose that $N = 589$ and $\lambda(N) = 90$. Now $589 \bmod 90 = 49$. Trying $\phi(N) = 540$, we get that the prime factors of N are the roots of the quadratic

$$x^2 - 50x + 589 = 0,$$

so that

$$p, q = 25 \pm \sqrt{625 - 589} = 25 \pm 6 = 31, 19.$$

There is no need to try the other case.

Example Suppose that $N = 21$ and $\lambda(N) = 6$. Then $N - \phi(N) = 3$ or 9. In the first case the quadratic is $x^2 - 4x + 21 = 0$, which has imaginary roots. In the second, it is $x^2 - 10x + 21 = 0$, with roots 3 and 7. Note that we only need the second case if $q - 1$ divides $p - 1$, since otherwise $\lambda(N) \geq 2(p-1)$.

Finally, we remark that, if $\phi(N)$ or $\lambda(N)$ is known, then problem (6) is easy. For we choose e to be the inverse of $d \bmod \lambda(N)$, using Euclid's Algorithm.

In the other direction, if we know a solution to problem (6) (that is, if we know d and e such that T_e is the inverse of $T_d \bmod N$), we can often factorise N. The algorithm is as follows. We assume that N is the product of two primes (neither of them being 2).

> Let $de - 1 = 2^a \cdot b$, where b is odd. Choose a random x with $0 < x < N$.
>
> First, calculate $\gcd(x, N)$. If this is not 1, we've found a factor already and we can stop.
>
> If $\gcd(x, N) = 1$, we proceed as follows. Let $y = x^b \bmod N$. If $y \equiv \pm 1 \pmod N$, the algorithm has failed. Repeatedly replace y by $y^2 \bmod N$ (remembering the preceding value of y – more formally, $z := y$ and $y := y^2 \bmod N$) until $y \equiv \pm 1 \pmod N$.
>
> If $y \equiv -1 \pmod N$, the algorithm has failed.
>
> However, if $y \equiv 1 \pmod N$, then we have found z such that $z^2 \equiv 1 \pmod N$ and $z \not\equiv \pm 1 \pmod N$. Then $\gcd(N, z+1)$ and $\gcd(N, z-1)$ are the prime factors of N.

Remarks:

- The chance that $\gcd(x, N) \neq 1$ is very remote. However, we should make this test, since the rest of the algorithm depends on the assumption that the gcd is 1.

- The loop where we do $z := y$ and $y := y^2 \bmod N$ will be repeated at most a times. For we know that $\lambda(N)$ divides de, so that $x^{de} \equiv x \pmod N$. Since x is coprime to N, it has an inverse, and so $x^{de-1} \equiv 1 \pmod N$. But $x^{de-1} = x^{2^a \cdot b} \equiv y^{2^a}$, where $x \equiv x^b$, so after a successive squarings we certainly have 1; the loop will terminate no later than this step.

- If $z^2 \equiv 1 \pmod N$, then N divides $z^2 - 1 = (z+1)(z-1)$. Both the factors lie between 1 and $N-1$, so $\gcd(N, z+1)$ and $\gcd(N, z-1)$ are proper divisors of N. They are coprime, so they must be the two prime factors of N.

- It can be shown that, choosing x randomly, the probability that the algorithm succeeds in factorising N is approximately $1/2$. So, by repeating a number of times with different random choices of x if necessary, we can be fairly sure of finding the factorisation of N.

Example Suppose that $N = 589$ and we are told that the private exponent corresponding to $d = 7$ is $e = 13$. Now $de - 1 = 90 = 2 \cdot 45$. Apply the algorithm with $x = 2$. We do have $\gcd(2, 589) = 1$. Now $y = 2^{45} \bmod 589 = 94$. At the next stage, $z = 94$ and $y = z^2 \bmod 589 = 1$. So the factors of 589 are $\gcd(589, 95) = 19$ and $\gcd(589, 93) = 31$ (these gcds are found by Euclid's algorithm).

Implementation

Bob chooses two large prime numbers p_B and q_B. This involves a certain amount of randomness. It is known that a fraction of about $1/(k \ln 10)$ of k-digit numbers are prime. Thus, if Bob repeatedly chooses a random k-digit number and tests it for primality, in mk trials the probability that he has failed to find a prime number is exponentially small (as a function of m). Each primality test takes only a polynomial number of steps. The chances of success at each trial can be doubled by the obvious step of choosing only odd numbers; and excluding other small prime divisors such as 3 improve the chances still further. We conclude that in a polynomial number of steps (in terms of k), Bob will have found two primes, with an exponentially small probability of failure.

Knowing p_B and q_B, Bob computes their product $N_B = p_B q_B$. He can also compute $\lambda(N_B) = \operatorname{lcm}(p_B - 1, q_B - 1)$. He now computes a large 'exponent' e_B satisfying $\gcd(e_B, k_B) = 1$ (again by choosing a random e and using Euclid's algorithm). The application of Euclid's algorithm also gives the inverse of e_B mod

$\lambda(N_B)$, that is the number d_B such that T_{d_B} is the inverse of T_{e_B}, where

$$T_{e_B} : x \mapsto x^{e_B} \pmod{N_B}.$$

Proposition 5.6 shows that the maps are inverses on all of $\mathbb{Z}/(N_B)$, since N_B is the product of two primes.

Bob publishes N_B and e_B, and keeps the factorisation of N_B and the number d_B secret.

If Alice wishes to send a message to Bob, she first transforms her message into a number x less than N_B. (For example, if the message is a binary string, break it into blocks of length k, where $2^k < N_B$, and regard each block as an integer in the interval $[0, 2^k - 1]$ written to the base 2. Now she computes $z = T_{e_B}(x)$ and sends this to Bob.

Bob deciphers the message by applying the inverse function T_{d_B} to it. This gives a number less than N_B and congruent to x mod N_B. Since x is also less than N_B, the resulting decryption is correct.

If Eve intercepts the message z, she has to compute $T_{d_B}(z)$, which is a hard problem (problem (6) above). Alternatively, she could compute d_B from the published value of e_B. Since d_B is the inverse of e_B mod $\lambda(N_B)$, this requires her to calculate $\lambda(N_B)$, which is also hard (problem (5)). Finally, she could try to factorise N_B: this, too, is hard (problem (4)). So the cipher is secure.

RSA signatures

Since the plaintext and ciphertext are both integers smaller than N_B, and the encryption function is a bijection, the RSA system supports digital signatures.

If Alice and Bob have both chosen a key, then Alice can sign her message to Bob by the method for digital signatures that we described earlier. That is, Alice 'decrypts' with her secret key T_{d_A} before encrypting with Bob's public key; after decrypting, Bob then 'encrypts' with Alice's public key to get the authenticated message.

Remark We saw that, if we know e and d such that T_d is the inverse of T_e mod N, then we have a very good chance of factorising N. The moral of this is that, if your RSA key is broken (that is, if Eve comes to know both e and d), it is not enough to keep the same N and choose different d and e, since you must assume that Eve now knows the factors of N. You must begin again with a different choice of two primes p and q.

5.3 Primes and factorisation

Primality testing is easy, but factorisation is hard. This assertion is the basis of the security of the RSA cipher. In this section, we consider it further.

Primality testing

The basic algorithm for primality testing, which you learn in Algorithmic Mathematics, is trial division. In a very crude form it asserts that, if $n > 1$ and n is not divisible by any integer smaller than $\sqrt{n}$, then n is prime.

The first thing to say about this algorithm is that, with minor modification, it leads to a factorisation of n into primes. If n is not prime, then the first divisor we find will be a prime p, and we continue dividing by p while this is possible to establish the exact power of p. The quotient is divisible only by primes greater than p, so we can continue the trial divisions from the point where we left off.

The second thing is that this simple algorithm does not run in polynomial time: the input is the string of digits of n, and the number of trial divisions is about $\sqrt{n}$, roughly $2^{k/2}$ if n has k digits to the base 2.

It is a bit surprising at first that primality testing can be easier than factorisation. This holds because there are algorithms which decide whether or not a number is prime without actually finding a factor if it is composite! Two examples of such theorems are:

Theorem 5.9 *Little Fermat Theorem: If n is prime then $x^n \equiv x \pmod{n}$ for any integer x.*

Wilson's Theorem: n is prime if and only if $(n-1)! \equiv -1 \pmod{n}$.

We have seen the proof of the little Fermat theorem. Here is Wilson's Theorem.

Suppose that p is prime. We know that every number x in the set $\{1,\ldots,p-1\}$ has an inverse y mod p (so that $xy \equiv 1 \pmod{p}$). The only numbers which are equal to their inverses are 1 and $p-1$: for if x is equal to its inverse, then $x^2 \equiv 1 \pmod{p}$, so that p divides $x^2-1 = (x-1)(x+1)$, and p must divide one of the factors. The other $p-3$ numbers in the range can be paired with their inverses, so that the product of each pair is congruent to 1 mod p. Now multiplying all these numbers together gives

$$(p-1)! \equiv 1 \cdot 1^{(p-3)/2} \cdot (p-1) \equiv -1 \pmod{p}.$$

Conversely, *suppose that p is composite*. If q is a prime divisor of p, then certainly q divides $(p-1)!$, and so $(p-1)!$ is congruent to a multiple of q mod p, and cannot be prime.

Can either of these results give us a quick test for primality?

As we explained in the last section, there is an efficient way to calculate $x^n \bmod n$, involving at most $2\log_2 n$ multiplications of numbers not exceeding n and calculation of the remainder mod n after each multiplication. Thus, for example, $2^{589} \equiv 326 \pmod{589}$, so we know that 589 is composite without finding any of its factors.

Unfortunately, this test doesn't work in the other direction. For example, $2^{341} \equiv 2 \pmod{341}$, even though $341 = 11 \cdot 31$ is not prime. We say that 341 is a pseudoprime to the base 2. In general, n is a *pseudoprime* to the base a if n is not prime but $a^n \equiv a \pmod{n}$. Pseudoprimes are not very common, and if we are prepared to take the small risk that the number we chose is a pseudoprime rather than a prime, then we could simply accept such numbers.

We could feel safer if we checked different values. For example, although 341 is a pseudoprime to base 2, we find that $3^{341} \equiv 168 \pmod{3}$, so that 341 is definitely composite.

Unfortunately even this does not give us complete confidence. A *Carmichael number* is defined to be a number that is a pseudoprime to every possible base but not a prime. It seems unlikely that such numbers exist, but they do!

Proposition 5.10 *The positive integer n is a Carmichael number if and only if it is composite and $\lambda(n)$ divides $n-1$.*

For a Carmichael number can have no repeated prime divisors: if p^2 divides n then $p^n \bmod n$ is a multiple of p^2 and so not equal to p. Now for such numbers we know that $x^m \equiv x \pmod{n}$ if and only if $\lambda(n)$ divides $m-1$.

Now consider $n = 561 = 3 \cdot 11 \cdot 17$. We have $\lambda(n) = \mathrm{lcm}(2,10,16) = 80$ and 80 divides 560, so 561 is a Carmichael number.

Refinements of this test due to Solovay and Strassen and to Rabin gave fast algorithms which could conclude either that n is certainly composite or that n is 'probably prime', where our degree of confidence could be made as close to 1 as required.

Wilson's theorem doesn't have the drawback of Fermat's Little Theorem: it is a necessary and sufficient condition for primality. That is, if $(n-1)! \equiv -1$

$(\mathrm{mod}\, n)$, then n is prime; if not, not. Unfortunately, unlike the situation of calcluating powers of integers, nobody has ever discovered a quick method of calculating factorials mod n for given n. (The natural method would require $n-1$ multiplications.)

The method finally used by Agrawal, Kayal and Saxena was a kind of combination of these two approaches, together with some ingenuity. They begin with the remark that n is prime if and only if

$$(x-1)^n \equiv x^n - 1 \pmod{n}$$

as *polynomials*, rather than integers. This is because the coefficients in $(x-1)^n$ are binomial coefficients $\binom{n}{i}$; if n is prime, then $\binom{n}{i}$ is a multiple of n for $i = 1,\ldots,n-1$, but if n has a prime factor q then $\binom{n}{q}$ is not a multiple of n.

This is no good as it stands; we can raise $x-1$ to the nth power with only $2\log_2 n$ multiplications, but the polynomials we have to deal with along the way have as many as n terms, too many to write down. So the trick is to work mod $(n, x^d - 1)$ for some carefully chosen number d. I refer to their paper for the details.

Factorisation

There is not a lot to say about factorisation: it is a hard problem! There are some special tricks which have been used to factorise huge numbers of some special forms, such as *Fermat numbers* $2^{2^n}+1$ and *Mersenne numbers* $2^p - 1$ (for p prime). Of course, we would avoid such special numbers when designing a cryptosystem.

However, one should not overestimate the difficulty of factorisation. Numbers with well over 100 digits can be factorised with today's technology. The gap between primality testing and factorisation is sufficiently narrow that it is necessary to keep updating an RSA system to use larger primes.

Later we may touch on quantum computing and see why the advent of this technology (if and when it comes) will allow efficient factorisation of large numbers and make the RSA system insecure.

We discuss briefly just one factorisation technique: *Pollard's $p-1$ method*. This method works well if the number N we are trying to factorise has a prime factor p such that $p-1$ has only small prime power divisors. Suppose that we can choose a number b such that every prime power divisor q of $p-1$ satisfies $q \leq b$.

The algorithm works as follows. Choose any number $a > 1$, and by successive powering compute $x = a^{b!} \bmod N$. By assumption, every prime power divisor of $p-1$ is at most b, and hence divides $b!$. Hence $p-1$ divides $b!$. Thus, $a^{b!} \equiv 1 \pmod{p}$ by Fermat's Little Theorem, so that p divides $x-1$. By assumption, p divides N. Hence $\gcd(x-1,N)$ is a multiple of p, and so is a non-trivial divisor of N. (Indeed, in the RSA case, if N is the product of two primes, then $\gcd(x-1,N)$ will be a prime factor of N.)

Here is an example. Let $N = 6824347$ and $b = 10$. Choosing $a = 2$, we find that $x = 5775612$ and $\gcd(x-1,N) = 2521$. Thus, 2521 is a factor of N, and with a bit more work we find that it is prime and that $N = 2521 \cdot 2707$ is the prime factorisation of N.

The method succeeds because

$$2521 - 1 = 2^3 \cdot 3^2 \cdot 5 \cdot 7$$

and all the prime power divisors are smaller than 10. Of course, if this condition were not satisfied, the method would probably fail. If we replace 2521 by 2531 in the above example, we find that $N = 2531 \cdot 2707 = 6851417$, $x = 2^{10!} \bmod N = 6414464$, and $\gcd(x-1,N) = 1$.

Because we have to calculate $a^{b!} \bmod N$ by successively replacing a by $a^i \bmod N$ for $i = 1,\ldots,b$, we have to perform $b-1$ exponentiations mod N. So the method will not be polynomial-time unless $b \leq (\log N)^k$ for some k. So we are only guaranteed success in polynomial time if the prime-power factors of $p-1$ for at least one of the divisors of N are at most $(\log N)^k$ – this is small compared to the magnitudes of the primes involved, which may be roughly $\sqrt{N}$.

Thus, in choosing the primes p and q for an RSA key, we should if possible avoid primes for which $p-1$ or $q-1$ have only small prime power divisors; these are the most vulnerable.

5.4 Diffie–Hellman key exchange

The functions used for the RSA cipher can also be used to implement the key-exchange protocol that we discussed at the very beginning of our discussion of public-key cryptography. This system of key exchange actually predates the RSA cipher.

Assume that Alice wants to send a secret message to Bob. Alice and Bob agree on a modulus p, a prime number. (They must share the prime p, so they must assume that Eve can get hold of it!) Each of them chooses a number coprime to

$\lambda(p) = p-1$, and computes its inverse. These numbers are not revealed. Alice chooses e_A and d_A, Bob chooses e_B and d_B. Note that our commutation condition is satisfied:

$$T_{e_A}T_{e_B}(x) = x^{e_Ae_B} \bmod p = T_{e_B}T_{e_A}(x).$$

In terms of our analogy, T_{e_A} is Alice putting on her padlock, while T_{d_A} is Alice removing her padlock.

Now Alice takes the message x and applies T_{e_A}; she sends $T_{e_A}(x)$ to Bob. Bob applies T_{e_B} and returns $T_{e_B}T_{e_A}(x)$ to Alice. Alice applies T_{d_A} and returns

$$T_{d_A}T_{e_B}T_{e_A}(x) = T_{d_A}T_{e_A}T_{e_B}(x) = T_{e_B}(x)$$

to Bob, who then applies T_{d_B} and recovers $T_{d_B}T_{e_B}(x) = x$, the original message.

Nobody has yet discovered a weakness in this protocol like the weakness we found using one-time pads. In other words, even if Eve intercepts all the messages $T_{e_A}(x)$, $T_{e_B}T_{e_A}(x)$ and $T_{e_B}(x)$ that pass to and fro between Alice and Bob, there is no known easy algorithm for her to discover x (even given the modulus p).

Contrast this with the standard RSA protocol: First, it allows a pair of users to communicate securely, whereas RSA allows any two users in a pool to communicate; secondly, three messages have to be sent, rather than just one; thirdly, what is secret and what is public are different in this case (the prime is public but the exponent is secret).

The security of this protocol depends on the fact that, if $y = x^e \pmod p$, then knowledge of x and y does not allow an easy calculation of e. For suppose that Eve could solve this problem. Recall that Eve knows x^{e_A}, x^{e_B} and $x^{e_Ae_B}$ (the three messages exchanged during the protocol). If she could use x^{e_A} and $x^{e_Ae_B}$ to discover e_B, she could find its inverse d_B modulo $p-1$ and then calculate $(x^{e_B})^{d_B} \bmod p = x$, the secret message.

Thus, the security of this method depends on the fact that the following problem is hard:

> Given x, y, and a prime p such that $y \equiv x^e \bmod p$, find e.

This is known as the *discrete logarithm problem*, since in a sense e is the logarithm of y to base x (where our calculations are in the integers mod p, rather than in the real numbers as usual). This problem is believed to be at least as difficult as factorisation, although (like factorisation) it is not known to be NP-complete.

If it happens that the order of x mod p is small (so that there are only a few distinct powers of x mod p), then e can be found by exhaustive search. So, to make

the problem hard, the order of x should be as large as possible. Ideally, choose x to be a primitive root mod p (an element of order $\lambda(p) = p-1$).

Example Suppose that $p = 30491$ and $x = 13$. Then $x^2 = 169$, $x^3 = 2197$, $x^4 = 28561$, and $x^5 \equiv 1 \pmod{p}$. So the discrete logarithm problem is easily solved. On the other hand, 2 is a primitive root mod 30491, so all the powers $2^0, 2^1, 2^2, \ldots, 2^{30489}$ are distinct, and finding which one is a particular element y will be very laborious.

How do we cheeck that 2 is a primitive root mod 30491, without actually working out all these powers? We know that $2^{30490} \equiv 1 \pmod{30491}$, by Fermat's Little Theorem. So the order of 2 must be a divisor of 30490. We factorise 30490 into prime factors: $30490 = 2 \cdot 5 \cdot 3049$. So any *proper* divisor would have to divide the product of two of these primes. So we check that none of $2^{2\cdot 5}$, $2^{2\cdot 3049}$ and $2^{5\cdot 3049}$ is congruent to 1 mod 30491. So in this case we only have to check three powers of 2; but it is necessary to factorise $p-1$.

5.5 El-Gamal

The El-Gamal cryptosystem is a rival to RSA and is widely used. Its ssecurity is based on the difficulty of the discrete logarithm. It works as follows.

Bob chooses a prime number p and a primitive root g mod p. (Remember that this is an element such that the powers $g^0, g^1, \ldots, g^{p-2}$ are all distinct modulo p, and include all the non-zero congruence classes mod p. We saw in Theorem 16 that primitive roots exist for any prime p.) He also chooses an integer $a \in \{1, \ldots, p-2\}$, and computes $h = g^a \pmod{p}$. His public key is (p, g, h); the number a is kept secret.

Alice wants to send a plaintext x to Bob, encoded as an integer in the range $\{1, \ldots, p-1\}$. She chooses a random number k, also in this range, and computes $y_1 = g^k \pmod{p}$ and $y_2 = xh^k \pmod{p}$. The ciphertext is the pair (y_1, y_2).

Note that

- the ciphertext is twice as long as the plaintext;
- there are $p-1$ different ciphertexts for each plaintext, one for each choice of the random number k.

Bob receives the message $(g^k, xh^k) \bmod p$. He knows the number a such that $h = g^a \bmod p$; so he can compute

$$h^k \equiv (g^a)^k \equiv (g^k)^a \bmod p$$

without knowing Alice's secret number k. Now he can find x by "dividing" $y_2 = xh^k$ by h^k; more precisely, he uses Euclid's algorithm to find the inverse of $h^k \bmod p$ and multiplies y_2 by this inverse to get the plaintext x.

Eve, intercepting the message, is faced with the problem of finding either

- the number a for which $h \equiv g^a \pmod p$, so that she can then use the same decrypting method as Bob; or
- the number k for which $y_1 \equiv g^k \pmod p$, so that she can find h^k directly and hence find x.

Either approach requires her to solve the Discrete Logarithm problem, and so may be assumed to be difficult. No better way of trying to break the cipher is known.

Note that, if Eve does have the computational resources to solve a discrete logarithm problem, she should employ them on the first of the above problems. For if this is solved, then she knows Bob's private key and can read all his mail. Solving the second only gives her Alice's random number k, which will be different for each message, so the same job would have to be done many times.

Here is a brief example. Suppose that Bob chooses the prime $p = 83$, the primitive root $g = 2$, and the number $a = 30$, so that $h = 2^{30} \bmod 83 = 40$. Bob's public key is $(83, 2, 40)$. Suppose that Alice's plaintext is $x = 54$ and her random number is $k = 13$. Then Alice's ciphertext is

$$(g^k, xh^k) \bmod p = (58, 71).$$

Bob computes $58^{30} \bmod 83 = 9$. By Euclid's algorithm, the inverse of 9 mod 83 is 37; and so the plaintext is $37 \cdot 71 \bmod 83 = 54$.

El-Gamal signatures

Using the El-Gamal scheme for digital signatures is a bit more complicated than using, say, RSA. This is because, as we saw, the ciphertext in El-Gamal is twice as long as the plaintext, and depends on the choice of a random number k. So, to sign a message, Alice cannot simply pretend that the message is a cipher and

decrypt it with her private key! Instead, she adds further data whose purpose is to authenticate the message.

Suppose that Alice's El-Gamal public key is (p,g,h), where p is prime and g is a primitive root mod p. Then $h \equiv g^a \pmod{p}$, where the number a is known only to Alice.

To sign a message $x \in \{1,\ldots,p-1\}$, Alice chooses a random number k satisfying $\gcd(k,p-1)=1$. Then using Euclid's algorithm, she computes the inverse l of k mod $p-1$. Now she computes

$$\begin{aligned} z_1 &= g^k \bmod p, \\ z_2 &= (x-az_1)l \bmod p-1 \end{aligned}$$

The signed message is (x,z_1,z_2). Just as in the case of encryption, note that it is longer than (in this case, three times as long as) the unsigned message, and it depends on a random number k. Alice then encrypts this message with Bob's public key and sends it to Bob.

On receipt, Bob decrypts the message, and finds three components. The first component is the plaintext x. The second and third components comprise the signature. Bob accepts the signature as valid if

$$h^{z_1} z_1^{z_2} \equiv g^x \pmod{p}.$$

We have to show that

- if Alice follows the protocol correctly, this condition will be satisfied;
- Eve cannot forge the signature (i.e. produce (x,z_1,z_2) satisfying this condition) without solving a discrete logarithm problem.

The first condition is just a case of checking;

$$h^{z_1} z_1^{z_2} \equiv g^{az_1} g^{kl(x-az_1)} \pmod{p}.$$

Note that $g^{p-1} \equiv 1 \pmod{p}$, so exponents of g can be read modulo $p-1$. Now $kl \equiv 1 \pmod{p-1}$, so $g^{kl(x-az_1)} \equiv g^{x-az_1} \pmod{p}$. Then

$$h^{z_1} z_1^{z_2} \equiv g^{az_1} g^{x-az_1} \equiv g^x \pmod{p}.$$

The second part is a bit more complicated and the argument will not be given here. It is clear that Eve cannot do Alice's computation without knowing a. We have to be sure that there is no other way that she could produce a forgery.

Example Suppose that Alice's public key is $(107, 2, 15)$, with secret number 11, so that 2 is a primitive root mod 107, and $2^{11} \equiv 15 \pmod{107}$. Suppose that Alice wants to send the message 10 to Bob and sign it. She chooses $k = 17$; this number is coprime to 106, and its inverse is 25. The signature is (z_1, z_2), where

$$\begin{aligned} z_1 &= 2^{17} \bmod 107 = 104, \\ z_2 &= (10 - 11 \cdot 104) \cdot 25 \bmod 106 = 58. \end{aligned}$$

So she encrypts the plaintext $(10, 104, 58)$ with Bob's public key and sends it to Bob. (Note that the one number x has now become six numbers in the ciphertext!)

Bob, having decrypted the message, obtains $(10, 104, 58)$. He tests whether

$$15^{104} \cdot 104^{58} \equiv 2^{10} \pmod{107},$$

and, since this is the case, he is assured that the message is from Alice.

5.6 Finding primitive roots

The El-Gamal system requires each user to choose a prime p and a primitive root g mod p. How does he find a primitive root? This is a problem which is itself not easy. There are two approaches that have been used.

One approach is to observe that it is not crucial for the operation of the method that g is a primitive root; all we require is that g should have many different powers mod p, so that the discrete logarithm cannot be solved by exhaustive search. So all that Bob has to do is to choose a number g and check that $g^i \not\equiv 1 \pmod{p}$ for all not-too-large i. (If he can factorise $p-1$, he can test whether g is a primitive root in only a few steps by the method of the earlier example; if it is not a primitive root, he can find out what its order actually is by continuing this analysis.)

Another is to observe that there are some special primes for which it is easy to find a primitive root. One way to do this is as follows.

A pair (q, p) of prime numbers is called a *Sophie Germain pair* if $p = 2q + 1$. These are so-called because, in 1825, Sophie Germain proved a special case of Fermat's Last Theorem for exponents which are the smaller of a Sophie Germain pair. The important thing is that it appears (though it is not proved yet) that there are lots of such prime pairs. So it is not too inefficient to find a prime q, and then test whether $p = 2q + 1$ is also prime.

Now we have the following result.

Proposition 5.11 *let (q,p) be a Sophie Germain pair. Suppose that $1<x<p-2$. Then x is a primitive root mod p if and only if $x^q \equiv -1 \pmod{p}$.*

For the order of x mod p divides $p-1=2q$ by Fermat's Little Theorem, and is not 1 or 2 (since the only elements with these orders are 1 and $p-1$); so the order is q or $2q$.

Suppose that x is a primitive root (of order $2q$), and let $y = x^q \bmod p$. Then $y^2 \equiv 1 \pmod{p}$, but $y \not\equiv 1 \pmod{p}$; so $x^q \equiv y \equiv -1 \pmod{p}$.

Conversely, suppose that x is not a primitive root; then x has order q, so $x^q \equiv 1 \pmod{p}$.

In our earlier example, $(41,83)$ is a Sophie Germain pair, so to test whether 2 is a primitive root mod 83, we only have to decide whether $2^{41} \equiv -1 \pmod{83}$. This can be done directly, but the calculation can be simplified still further using tools from Number Theory (the so-called Quadratic Reciprocity Law of Gauss). This is beyond the scope of this course, but is discussed in the Number Theory course.

Sophie Germain was the first female mathematician in western Europe. She faced many difficulties in being accepted as a serious mathematician. She communicated by letter with many of the famous mathematicians of the time, such as Gauss and Lagrange, signing her name "Monsieur LeBlanc". Gauss learned that his correspondent was a woman in a curious way.

He lived in Braunschweig in eastern Germany. When Napoleon's armies invaded in 1806, Germain asked the military commander, who was a family friend, to take special care of Gauss. (As a child, she had read the story of how Archimedes had been killed by a Roman soldier during the invasion of Syracuse, and dreaded that Gauss would suffer the same fate.) On asking to whom this special attention was due, Gauss was surprised to learn that "Monsieur LeBlanc" was a woman.

Exercises

5.1. Bob's RSA public key is $(8633, 151)$.

(a) Encrypt the plaintext 1000 for transmission to Bob.

(b) Factorise 8633.

(c) Decrypt the ciphertext 8119 which was sent to Bob.

(d) Sign the text 5000 for Bob.

5.2. This question shows how sometimes we can restrict the prime divisors of numbers of special form.

Let F_n be the *Fermat number* $2^{2^n}+1$. Suppose that p is a prime divisor of F_n.

(a) Show that the order of 2 mod p is 2^{n+1}.

(b) Deduce that $p \equiv 1 \pmod{2^{n+1}}$.

(c) Find a prime divisor of F_5.

(d) Show (without actually doing the calculations) that this prime divisor of F_5 could be found by Pollard's $p-1$ method, taking $a=3$ and $b=8$.

5.3. (a) Show that 91 is a pseudoprime to base 3.

(b) Show that 1105 is a Carmichael number.

5.4. Bob's El-Gamal public key is $(53,2,3)$,

(a) Encrypt the plaintext 10 for transmission to Bob.

(b) What is Bob's secret number?

(c) Decrypt the ciphertext $(44,45)$ which was sent to Bob.

(d) Sign the text 30 for Bob, and verify that the signature is valid.

Chapter 6

Secret sharing and other protocols

In this chapter we will see two things: protocols for purposes similar to sending secret messages but with variations; and other kinds of attacks on cipher systems and how they might be analysed.

6.1 Secret sharing

The President of the Commercial Bank of Nod is the only person who holds a secret password which opens the bank vault. He realises that he can't always be around, and sometimes it is necessary to open the vault in his absence. But he doesn't trust any of his employees with the password. So he wants to give each of the two Vice-Presidents of the bank some partial information, so that only if the two of them combine their information can they open the vault. How can he do this?

He could simply give half the password to each Vice-President. But then there is a risk that one of the Vice-Presidents will guess the other half of the password: this is much easier than guessing the whole password. He wants a method where the information given to each Vice-President is no help to him in guessing the password on his own.

This is not difficult. He can simply encrypt the password, and give one Vice-President the ciphertext and the other the key. Together they can decrypt the password and open the vault. But if the cipher is a secure one such as a one-time pad, one Vice-President alone cannot break it, or even get any information about it (by Shannon's Theorem).

Slightly more formally, suppose that the password is a string over an alphabet

A with q symbols. Let L be a $q \times q$ Latin square whose rows and columns are labelled by A, and whose entries are symbols from A. Suppose that $z = z_1 \dots z_n$ is the password. Choose any random string $a = a_1 \dots a_n$ of symbols of A, and let $b = b_1 \dots b_n$ be the string for which $a_i \oplus b_i = z_i$ for $i = 1, \dots, n$, where $s \oplus t$ is the symbol in row s and column t of the square. (This is slightly different from the way we did it before but the difference is immaterial.)

As usual, we write $z = a \oplus b$ to mean coordinatewise operation, that is, $z_i = a_i \oplus b_i$ for $i = 1, \dots, n$.

For example, let $A = \{0, \dots, q-1\}$ be the set of integers mod q, and L is the addition table mod q (so that $s \oplus t = s + t \bmod q$).

This example can easily be extended to the case where there are k Vice-Presidents, and it is required that only all k acting together can open the vault. Let us suppose that the Latin square is the addition table mod q. In this case, the jth Vice-President is given the information $a^{(j)} = a_1^{(j)} \dots a_n^{(j)}$, where

$$z = a^{(1)} \oplus a^{(2)} \oplus \cdots \oplus a^{(k)}.$$

(For an arbitrary Latin square the method is the same, but we have to be careful about the order we do the additions.)

Not only is it true that any $k-1$ of the Vice-Presidents cannot work out the password; they cannot get any information at all about it. For example, suppose that the first $k-1$ Vice-Presidents co-operate. They can calculate

$$b = a^{(1)} \oplus a^{(2)} \oplus \cdots \oplus a^{(k-1)}.$$

Now

$$b \oplus a^{(k)} = z,$$

but because of the Latin square property, in each row every symbol occurs once, so without knowledge of $a^{(k)}$ all strings are equally likely!

We can extend this idea still further with a definition as follows. Let k and t be positive integers with $k > t$, and let A be an alphabet of q symbols. A (k,t) *orthogonal array* over q is defined to be an array M with k rows and q^t columns with entries from A, having the following property:

> Given any t rows of M, and any t elements $a_1, \dots, a_t$ of A, there is exactly one column of A in which the entries $a_1, \dots, a_t$ occur (in that order) in the t chosen rows.

The numbers k and t are called the *degree* and *strength* respectively of the orthogonal array. The number of columns must be q^t, since this is the number of choices of a t-tuple $(a_1,\ldots,a_t)$.

Recall that, for a Latin square with symbol set $\{1,\ldots,n\}$, we constructed an array with three rows and q^2 columns, where the three entries of each column give the row number, column number, and symbol contained in a cell of the square. The defining properties of a Latin square translate into the fact that this is an orthogonal array of degree 3 and strength 2. (A row and column uniquely determine a symbol; a row and symbol uniquely determine a column; and a column and symbol uniquely determine a row.)

A (k,t) *secret sharing scheme* is a scheme in which each of k individuals is given a member of a set S in such a way that any t of the individuals acting together can determine the identity of a secret member s of S, but no $t-1$ individuals can get any information about s.

Theorem 6.1 *From an orthogonal array of degree k and strength t over A, we can construct a $(k-1,t)$ secret sharing scheme over the set A^n of strings of length n of elements of A.*

The construction works as follows. We can take $n=1$, since to "encode" a string we simply deal with its characters one at a time.

Suppose that M is an orthogonal array of degree k and strength t over A. The array M is regarded as public.

Now M has q^t columns. Exactly q^{t-1} of these have the property that the secret s appears in the last row. Choose one of these columns at random, and give the entry in its ith row to the ith individual in the secret-sharing scheme for $i=1,\ldots,k-1$.

Now by the properties of an orthogonal array, any t of the individuals can, by pooling their information, determine the chosen column of M, and hence its last entry, which is the secret. However, the information held by any $t-1$ individuals only determines a set of q columns, with the property that each symbol occurs in the last row of precisely one of these columns. So the individuals concerned can obtain no information about the secret.

Thus, a Latin square gives a $(2,2)$ secret sharing scheme, and we have seen that we can use it to construct a (k,k) secret sharing scheme for any k.

Orthogonal arrays with higher strength are a bit harder to construct. Here is a very nice construction due to K. A. Bush:

Theorem 6.2 *Let q be a prime power, and t a positive integer less than $q+1$. Then there exists an orthogonal array of degree $q+1$ and strength t over an alphabet of size q (and hence a (q,t) secret sharing scheme over an alphabet of size q).*

In this case, the alphabet is the finite field $\mathrm{GF}(q)$ with q elements. The array is constructed as follows.

Consider polynomials of degree $t-1$. Each such polynomial has the form

$$f(x) = a_0 + a_1 x + \cdots + a_{t-1}x^{t-1},$$

where $a_0, a_1, \ldots, a_{t-1} \in \mathrm{GF}(q)$. So there are q choices for each of the t coefficients, and hence q^t polynomials.

From any polynomial $f(x)$, we construct a column of length $q+1$ as follows. If the elements of $\mathrm{GF}(q)$ are numbered $u_1, \ldots, u_q$, we put $f(u_i)$ in the ith row, for $i = 1, \ldots, q$. In the $(q+1)$st row, we put the leading coefficient a_{t-1} of $f(x)$.

This gives an array with $q+1$ rows and q^t columns. It remains to show that it is an orthogonal array of strength t. Suppose we seek a column in which rows $i_1, \ldots, i_t$ contain entries $z_1, \ldots, z_t$ respectively.

Suppose first that none of these rows is the $(q+1)$st. To ease notation, we put $u_{i_j} = v_j$ for $j = 1, \ldots, t$. Then we have to show that there is a unique polynomial $f(x)$ of degree at most $t-1$ such that it takes prescribed values at t given points, namely

$$f(v_j) = z_j, \qquad j = 1, \ldots, t.$$

This is true in general; the method for finding the polynomial is known as *Lagrange interpolation*. In the case of a finite field, it can be proved by simple counting. We give this argument, and then the general proof (which has the advantage of being constructive).

First, we observe that there is at most one polynomial of degree $\leq t-1$ taking these values. For if f and g were two such polynomials, then $f-g$ would be zero at each point $v_1, \ldots, v_t$, contradicting the fact that a polynomial cannot have more roots than its degree. (This part of the argument works for any field.)

Now, there are q^t choices of the t values $z_1, \ldots, z_t$, and there are q^t choices of the coefficients of the polynomial

$$f(x) = a_0 + a_1 x + \cdots + a_{t-1}x^{t-1},$$

so each list of values must be realised by a polynomial.

The constructive method works as follows. Let

$$g_i(x) = \prod_{j \neq i} \frac{(x - v_j)}{(v_i - v_j)}.$$

Then we have $g_i(v_i) = 1$, and $g_i(v_j) = 0$ for $j \neq i$. Hence

$$f(x) = \sum_{i=1}^{t} z_i g_i(x)$$

satisfies $f(v_i) = z_i$ for $i = 1, \ldots, t$.

Now suppose that one of the rows (say the last) is the $(q+1)$st. Then in place of what went before, the last equation is now $a_{t-1} = z_t$. This equation determines a_{t-1}, and so we have to interpolate a polynomial h of degree $\leq t-2$ taking the other $t-1$ values

$$h(v_i) = z_i - a_{t-1} v_i^{t-1}, \qquad i = 1, \ldots, t-1.$$

By the same argument as before, there is a unique such polynomial.

The implementation of this secret-sharing scheme is remarkably simple. The President takes the secret password to be the coefficient of x^{t-1} in the polynomial, and chooses the coefficients of lower-degree terms at random. Then he evaluates the polynomial on the elements of the field, and gives one value to each Vice-President.

Any t of the Vice-Presidents can now use Lagrange interpolation, as described above, to find the unique polynomial of degree at most $t-1$ taking the values they have been given. Its leading coefficient is the secret. On the other hand, fewer than t Vice-Presidents can gain no information at all about the secret.

Example Figure 6.1 is the orthogonal array of degree 4 and strength 3 constructed by the above method. I have transposed the array for convenience in printing. We take all polynomials of degree at most 2 over $\mathrm{GF}(3) = \{0, 1, 2\}$. The components of the 4-tuple are $f(0)$, $f(1)$, $f(2)$, and the coefficient of x^2 in $f(x)$

Run your fingers down any three columns of the array on the right, and you should find that each of the $3^3 = 27$ possible triples occur exactly once.

Remark Bush's orthogonal arrays are known, in different terminology, as *Reed-Solomon codes*, and are used for error correction in CD players.

Polynomial	4-tuple
0	0000
1	1110
2	2220
x	0120
$x + 1$	1200
$x + 2$	2010
$2x$	0210
$2x + 1$	1020
$2x + 2$	2100
x^2	0111
$x^2 + 1$	1221
$x^2 + 2$	2001
$x^2 + x$	0201
$x^2 + x + 1$	1011
$x^2 + x + 2$	2121
$x^2 + 2x$	0021
$x^2 + 2x + 1$	1101
$x^2 + 2x + 2$	2211
$2x^2$	0222
$2x^2 + 1$	1002
$2x^2 + 2$	2112
$2x^2 + x$	0012
$2x^2 + x + 1$	1122
$2x^2 + x + 2$	2202
$2x^2 + 2x$	0102
$2x^2 + 2x + 1$	1212
$2x^2 + 2x + 2$	2022

Figure 6.1: An orthogonal array

6.2 Other protocols

Session keys

Public-key cryptography is slower than cryptography based on a shared secret key. So many systems, including PGP, have an initial round where a public-key cipher is used to share a secret key between the two participants of a session. The key is used only for that communication session.

The simplest way to do this is a modification of the Diffie–Hellman method. It has the advantage that the key itself is not transmitted, even in enciphered form.

Alice and Bob share a prime number p and a primitive root g mod p. (They must assume that Eve knows p and g as well.) Now Alice choses a number a in the range $\{0,\ldots,p-2\}$ and Bob chooses b in the same range. Alice computes $g^a \bmod p$ and sends it to Bob; Bob computes $g^b \bmod p$ and sends it to Alice. Now each of them can compute $(g^a)^b = (g^b)^a \bmod p$; this is the session key.

To obtain the key, Eve knows g^a and g^b, but needs either a or b to proceed further; so she needs to solve a discrete logarithm problem. Since a new key can be chosen for each session, Eve cannot pre-compute the discrete logarithm of a public key as in the case of El-Gamal.

Note that, as opposed to the protocol described before, this method requires only two, rather than three, transmissions, and these are asynchronous (that is, they can occur in either order).

What else?

Protocols for many other tasks have been derived. For example, Alice can send Bob a message which he has a 50% chance of being able to decrypt, and Alice herself doesn't know whether or not Bob can decrypt it. Similarly, she can send him a message which allows him to learn one or other of two secrets, so that Alice does not know which secret Bob has learned. Bob may construct a smart card which knows his secret key, and can prove that it knows it, but without revealing the secret key.

Fanciful as these may sound, they have been suggested to solve real practical problems. The last protocol, for example, has been proposed by Shamir as the basis for an electronic passport.

6.3 Other kinds of attack

For the most part, we have imagined Eve as just a snooper who intercepts a message from Alice to Bob and must be prevented from knowing its contents. There are more active roles that she can play. Here are a few examples. There has been a lot of work on deciding whether the ciphers we have discussed are secure against this kind of attack. Sometimes we must imagine that Eve is someone who works in Alice's or Bob's organisation, or someone who has complete control of the communication channel between them.

- Eve may have access to some ciphertexts from Alice to Bob together with the corresponding plaintext. Does this help her break future messages?

- Eve may, in some circumstances, be able to persuade Alice to encipher messages of Eve's choosing. Carefully-chosen messages may give more information than arbitrary messages.

- Eve may be able to impersonate Alice to a greater or lesser degree. For example, she can certainly send Bob a message claimimg to come from Alice, encrypted with Bob's public key. Alice can foil this by signing or authenticating her messages; we have seen how to do this in both RSA and El-Gamal. Even in this case, Eve may be able to send Bob some previously-intercepted ciphertexts instead ot the current ciphertext that Alice wants to send.

- Alice may later wish to repudiate a message she has sent to Bob, claiming that it was a forgery from Eve. If it is signed (and the signature includes a date and time), this should not be possible; but it seems difficult to prevent Alice from claiming that her private key has been obtained illicitly by Eve.

6.4 Social issues

Now that more-or-less unbreakable encryption is available to all, we have to ask: Do we value this privacy more than we fear that criminals, terrorists and others will be able to profit from it? There is a serious clash here between the ideal of civil liberty and the desire for law and order.

Some governments such as that of the USA have attempted to limit the complexity of the ciphers used by their citizens so that the intelligence and law-

enforcement agencies can read their communications. In an increasingly globalised society this is difficult to implement; in the long run it is probably doomed to failure. In any case, if a citizen sends messages using some unbreakable cipher, the authorities will be aware of this and will investigate the citizen and his correspondents by more conventional methods!

Chapter 7

Quantum effects

> There was a time when the newspapers said that only twelve men understood the theory of relativity. I do not believe there ever was such a time . . . On the other hand, I think I can safely say that nobody understands quantum mechanics.
>
> Richard Feynman,
> *The Character of Physical Law*

In this final chapter we consider some very recent developments based on the mysteries of quantum theory. I cannot attempt to explain these mysteries (and I needn't be ashamed to say that I don't understand them myself), but I have tried to lay out what quantum theory has to say about the behaviour of subatomic systems, and how this behaviour is relevant to cryptography.

There are two aspects which we treat in turn. First, the possibility of building a quantum computer has been raised. Such a gadget could efficiently solve the hard problems on which modern public-key cryptography depends (factorisation and discrete logarithm). Second, a cryptosystem has been proposed which allows Alice and Bob to detect if their communication has been compromised before any secret plaintext is entrusted to the communication channel.

7.1 Quantum basics

Like any physical theory, the purpose of quantum mechanics is to predict the result of a measurement on a physical system. But unlike all other theories, it

does not usually predict a single value, but offers only a probabilistic prediction, along the lines "the electron's spin will be in the direction of the magnetic field with probability $\frac{1}{2}$, and will be in the opposite direction with probability $\frac{1}{2}$".

At the same time, the system is affected by the measurement; the action of measurement changes the state of the system into one which depends on the result of the measurement.

We turn these principles into a more mathematical format. According to quantum mechanics, the state of a physical system is described by a unit vector in a certain complex inner product space (more precisely, a Hilbert space) called the *state space*, whose dimension may be finite or infinite depending on the system being considered.

An unobserved system "evolves" by what might be regarded as a rotation of the state space. More precisely, a system in state v at a certain time is in state Uv at some later time, where U is a *unitary* transformation (this means that $U^{-1} = \overline{U}^{\top}$, where the bar denotes complex conjugation. The exact form of U is determined by the laws of quantum mechanics (the Schrödinger equation).

However, when we make a measurement on the system, something different happens. A measurement is described by a *Hermitian* transformation H of the state space (one satisfying $H = \overline{H}^{\top}$). Now a standard theorem of linear algebra says that, if H is Hermitian, then the space has an orthonormal basis consisting of eigenvectors of H. We assume for simplicity that the eigenvalues of H are all distinct, so that $He = \lambda e$ holds for a one-dimensional space of eigenvectors e (given the eigenvalue λ). Now the laws of quantum mechanics state the following:

- The result of a measurement associated with H is an eigenvalue λ of H.
- If the system was in state v before the measurement, where $v = \sum a_\lambda e_\lambda$ is the expression for v in terms of an orthonormal basis of eigenvectors, then the probability that the result of the measurement is λ is $|a_\lambda|^2$. (These probabilities sum to 1 because v is a unit vector.)
- If the result of the measurement is λ, then immediately after the measurement the state of the system has "jumped" to e_λ.

Another theorem of linear algebra asserts that the eigenvalues of a Hermitian transformation are real numbers. This corresponds to the statement that the result of any physical measurement is a real number, even though the formalism uses vector spaces over the complex numbers.

7.2 Quantum computing

The standard systems considered in a quantum theory course, such as the hydrogen atom, have infinite-dimensional state spaces. However, to describe how to deal with a single bit of information quantum-mechanically, we only need a two-dimensional state space, whose basis vectors describe the two possible results of measuring the bit.

Thus, a *qubit* (short for "quantum bit") is a system whose state space is two-dimensional, spanned by the vectors e_0 and e_1. The operator H associated with the measurement of the bit is

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

relative to this basis. Thus $He_0 = 0$ and $He_1 = e_1$. So the eigenvalues of H are 0 and 1, and the corresponding eigenvectors are e_0 and e_1.

A typical state of the system (a unit vector in this space) has the form $ae_0 + be_1$, where a and b are complex numbers satisfying $|a|^2 + |b|^2 = 1$. If the system is in this state, we regard it as being in a *superposition* of the states e_0 (bit value 0) and e_1 (bit value 1). If we measure the value of the bit, we find that the probability that it is zero is $|a|^2$, while the probability that it is one is $|b|^2$.

The matrix

$$U = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

is unitary. It satisfies $Ue_0 = (e_0 + e_1)/\sqrt{2}$ and $Ue_1 = (e_0 - e_1)/\sqrt{2}$. Suppose that we have a circuit whose effect on a qubit (in one unit of time) is to apply U to the state vector. If we prepare the system with the bit taking a definite value, either 0 or 1, then one time unit later the bit is "smeared out" between the two states, that is, the result of a measurement will be 0 with probability $\frac{1}{2}$, and 1 with probability $\frac{1}{2}$. Since the equations are linear, the subsequent evolution of the system will be a superposition of the two states describing the evolution starting from a value 0 and from a value 1. In other words, the computer can perform two computations simultaneously!

The circuit which realizes U is called a *Hadamard gate*.

More generally, an n-qubit system has state space which has a basis consisting of unit vectors e_s, where s runs over all 2^n possible binary strings of length n. If we set up the system with each qubit taking a definite value, and then pass each one through a Hadamard gate, the resulting state will be an equal superposition

of all 2^n possible states, and we have a computer which can do 2^n calculations at once.

This is the basis of the power of a quantum computer. In very rough terms: with n qubits at our disposal, we can regard the 2^n strings as representing the integers $1, \ldots, 2^n$, and we can do trial divisions of N by all these numbers simultaneously, arranging the circuitry so that only values which divide exactly give rise to an output. Thus, we can factorise numbers as large as 2^{2n} with such a machine.

This is a rough description of *Shor's algorithm*, which uses a quantum computer to factorise large numbers efficiently. Space does not allow a more precise description.

Other tasks which quantum computers can do very quickly include sorting, and solving the discrete logarithm problem. We see that neither RSA nor ElGamal will be secure if a practical quantum computer is ever built.

The theory of quantum computing is well understood. The difficulties now are, in some sense, only technological ones. However, they are very severe. Most obviously, a quantum computer uses a single electron or atomic nucleus to store one qubit of information. (For example, as we saw earlier, if an electron is in a magnetic field, then a measurement of its spin will be either in the direction of the magnetic field or in the opposite direction, and we can take these two states as e_0 and e_1.) Now a single electron is very sensitive to interference from a cosmic ray or from thermal agitation by its surroundings. Thus, errors creep in at a very high rate.

By contrast, a bit in a classical computer is stored in a transistor where the difference between "charged" and "discharged" is of the order of trillions of electrons. A cosmic ray may eject a few of these electrons without affecting the bit. Classical computers are extremely reliable and fault-tolerant.

7.3 Quantum cryptography

In this section we will see how one of the key properties of quantum theory, that a measurement changes the state of the system, can be used to produce a "tamper-proof" cipher, where Alice and Bob can tell (with probability arbitrarily close to 1) whether Eve has been intercepting their communication, before any plaintext is actually sent.

The cryptosystem uses photons as opposed to electrons. These are the quanta of the electromagnetic field, and except in "photon traps" in cutting-edge research labs, they go their way at the speed of light, so are ideal for transmitting messages

but useless for computation. Some properties of photons which we will use are:

(i) A photon has a polarisation, in a direction perpendicular to the direction of travel. (Think of it as like a wave vibrating in a direction perpendicular to the direction of travel. This is really a simplification, since in fact a photon can have two vibrations superimposed, but it is good enough for the argument here.) Note that, for example, "up" and "down" describe the same polarised state.

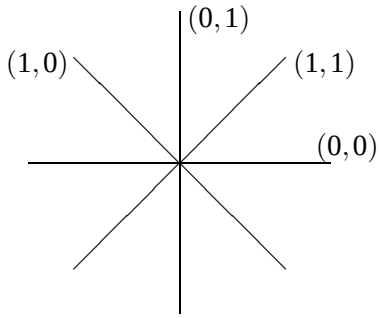
(ii) It is possible to prepare a photon which is polarised in any prescribed direction.

(iii) We can measure the polarisation in any direction; the answer to our measurement will be either "yes" or "no". If the actual polarisation direction makes an angle θ with the direction of the measurement, then the answer "yes" will be obtained with probability $\cos^2\theta$, and "no" with probability $\sin^2\theta$; these sum to 1, as probabilities should. Note that measurements in two perpendicular directions give exactly the same information. In particular, then, if we measure in the direction of the actual polarisation, we certainly get the answer "yes" (as $\cos 0 = 1$); and if we measure perpendicular to the actual polarisation, we get the answer "no" (as $\cos \pi/2 = 0$). In any other case, the result is random.

(iv) After the measurement, if the result was "yes", then the photon will be polarised in the direction of the measurement; if the result was "no", it will be polarised in the perpendicular direction.

The cryptosystem now works as follows. Alice and Bob use quantum effects to share a random sequence of bits, which they then use as a conventional one-time pad. We assume that all channels of communication between them are tapped by Eve.

Stage 1: Alice chooses independently two random binary sequences of length N, say $a_1a_2\ldots a_N$ and $b_1b_2\ldots b_N$. The number N should be a bit more than twice as long as the length of the plaintext bitstring, as we will see. For $i = 1,\ldots,N$, she prepares a photon whose state of polarisation is given in the following diagram, depending on (a_i, b_i). (The direction of travel is perpendicular to the paper, and the angles between adjacent lines are $\pi/4$.)



Note that a_i determines the choice of "orthogonal" (horizontal and vertical) or "diagonal" axes, and b_i determines which of the two axes to use.

Bob chooses a random binary sequence of length N, say $c_1 c_2 \ldots c_N$ (before the photons are sent). Now, if $c_i = 0$, he measures the polarisation of the ith photon in the vertical (or equivalently the horizontal) direction, and defines $d_i = 0$ if he finds that the polarisation is horizontal and $d_i = 1$ if it is vertical. On the other hand, if $c_i = 1$, then he measures the polarisation of the ith photon in one of the diagonal directions (again, the two measurements are equivalent, so he can make either), and sets $d_i = 0$ if he finds the polarisation to be in the NW–SE direction, and $d_i = 1$ if it is in the NE–SW direction.

Note that

- if $a_i = c_i$, then $b_i = d_i$;
- if $a_i \neq c_i$, then d_i is random: $P(d_i = b_i) = P(d_i \neq b_i) = \frac{1}{2}$. For in this case, Bob's measurement is at an angle of $\pi/4$ or $3\pi/4$ to the actual polarisation, and $\cos^2\theta = \sin^2\theta = \frac{1}{2}$ if $\theta = \pi/4$ or $\theta = 3\pi/4$.

Stage 2: Now Alice and Bob communicate in the ordinary way (over a line which might be insecure). Alice reads out her sequence $a_1 \ldots a_N$, and Bob reads out his sequence $c_1 \ldots c_N$. Since the sequences are both random, the number of places where they agree will be a binomial random variable $\mathrm{Bin}(N, \frac{1}{2})$, with mean $N/2$ and variance $N/4$ (that is, standard deviation $\sqrt{N}/2$); so it is very likely that the number lies in the range $N/2 \pm c\sqrt{N}$ for some moderate constant c. In this situation, we will say "the sequences agree at about $N/2$ places".

Stage 3: Now Alice and Bob discard the terms of their sequences $b_1 \ldots b_N$ and $d_1 \ldots d_N$ apart from those where the a and c sequences agree. They use what remains as a one-time pad. Since it is a subsequence of Alice's original random

sequence $b_1 \dots b_N$, it is a random sequence, of length about $N/2$. By Shannon's Theorem, their communication will be secure.

Note that $3N$ random bits have to be chosen in order to produce a shared key of length about $N/2$: this is in a sense the price paid for secrecy.

How could Eve attack this cipher?

If she uses any information she gains in stages 2 and 3, she will only be able to obtain about half of the one-time pad, which is no better than guessing randomly. For although she knows which subsequence of the original sequence will be used, she does not know the contents of this subsequence, since Alice and Bob do not reveal the b and d sequences at this stage.

What if Eve intercepts the photons? She can measure the polarisations, and then either let these photons continue their journey to Bob, or replace them with new photons whose polarisation is hers to choose. We show that, not only Eve cannot get hold of more than half of the key even in this way, but that Alice and Bob can detect her tampering. I will just consider the case where she sends the photons on to Bob after measuring the polarisations.

Eve must set up detectors according to some binary sequence $e_1 \dots e_N$, just as Bob does. Her sequence may be random or determinate: for example, she might set them all horizontally. But her choices will agree with Alice's random choices about half the time, and with Bob's about half the time, independently. So she can only be sure of getting about $N/4$ bits of the one-time pad.

To see how we detect tampering, note that if Eve chooses $e_i = a_i$, then she does not change the state of the photon and so her interference is undetectable. However, if she chooses $e_i \neq a_i$, and if $c_i = a_i$, then Alice and Bob have an even chance of detecting the interference. For suppose that $a_i = c_i = 0$ and $e_i = 1$. Then Eve changes the polarisation of the photon from orthogonal to diagonal (each of the two diagonals having probability $\frac{1}{2}$. For each possible state, Bob has probability $\frac{1}{2}$ of measuring horizontal polarisation, and $\frac{1}{2}$ of measuring vertical polarisation. So the probability that he measures the opposite of what Alice sent is $\frac{1}{2} \cdot \frac{1}{2} + \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{2}$.

Now Alice and Bob adopt the following procedure. They choose their sequences (a_i), (b_i) and (c_i) of length $N + 2n$ rather than N (where n is to be specified later). By Stage 2, they have agreed on about $N/2 + n$ positions where their sequences (b_i) and (d_i) will agree, if there has been no eavesdropping. Alice chooses n positions at random from this subsequence, and reveals their contents to Bob. If there is no eavesdropping, then Bob will have exactly the same bits in these positions as Alice. However, if Eve has been at work, the probability that Bob's bit disagrees with Alice's in one of these positions is $\frac{1}{4}$ (since this requires

that $e_i \neq a_i$ and that the randomness in quantum theory produces a result different from what was sent, each of which independently has probability $\frac{1}{2}$). So the probability that Alice and Bob are in complete agreement on the bits Alice reads out is only $(3/4)^n$.

This probability can be made arbitrarily small by choosing n large enough. For example, if $n = 73$, then $(3/4)^n < 1/10^9$, so the chance that Eve's interference is undetected is less than one in a billion. Increasing this to $n = 241$ would reduce the chance to less than one in 10^{30}.

Chapter 8

Bibliography

There are many books on cryptography. The list here includes only those books which I have consulted while preparing the lectures or course material for the course, or for the examples or quotations in the text.

Singh's book is an excellent and highly recommended introduction to cryptography ancient and modern, with detours about such topics as the decipherment of ancient scripts (Egyptian hieroglyphics and Linear B). Churchhouse's book is also introductory, and gives a wealth of detail and exercises on 20th century cipher machines such as Enigma and Hagelin.

Two fictional accounts of breaking a substitution cipher are "The Gold-Bug", by Edgar Allan Poe, and the Sherlock Holmes story "The Adventure of the Dancing Men", by Sir Arthur Conan Doyle. The two novels not containing the letter `a` are *Gadsby*, by Ernest Vincent Wright, and *A Void*, by Georges Perec (translated by Gilbert Adair). Wright's novel is hard to obtain now, but the text can be found at `http://gadsby.hypermart.net/`.

Dorothy L. Sayers, in *Have His Carcase*, gives a carefully worked example of breaking a Playfair cipher, using a short crib (a guessed portion of plaintext, in this case a date).

Babbage's breaking of the Vigenère cipher is treated briefly in his biography by Swade (as well as in Singh's book). Gaines' book, written in 1939, has a wealth of detail on cryptography before its mechanisation, including frequency tables, and many examples and exercises.

Garrett's and Stinson's books are textbooks for the mathematically inclined. Stinson's covers Shannon's Theorem, the two most popular public-key systems, hashing, signatures, and the Data Encryption Standard. Garrett's book gives te background from algebra, number theory, probability, etc., in separate chapters.

The Bletchley Park cryptanalysis is the subject of the books by Welchman (concentrating on Enigma) and Hinsley and Stripp (who range more widely). Hodges' biography of Alan Turing also includes a lot of detail on Enigma. Marks describes vividly the codes used by Allied secret agents in Europe.

The best textbook on quantum information theory is the book by Nielsen and Chuang. See also John Preskill's Caltech notes on quantum computing, at `http://www.theory.caltech.edu/people/preskill/ph219/`. Garey and Johnson is a standard work on computational complexity.

The proof by Agrawal *et al.* that primality testing is polynomial-time solvable is at `http://www.cse.iitk.ac.in/news/primality.html`.

Anon, *The Mabinogion* (transl. Jeffrey Gantz), Penguin Books, London, 1976.

Robert Churchhouse, *Codes and Ciphers: Julius Caesar, the Enigma, and the Internet*, Cambridge University Press, Cambridge, 2002.

Sir Arthur Conan Doyle, *The Complete Sherlock Holmes*, Penguin (reprint), London, 1981.

Richard Feynman, *The Character of Physical Law*, BBC publications, London, 1965.

Helen Fouché Gaines, *Cryptanalysis: A Study of Ciphers and their Solution*, Dover Publ. (reprint), New York, 1956.

M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, 1979.

Paul Garrett, *Making, Breaking Codes: An Introduction to Cryptology*, Prentice-Hall, Upper Saddle River, 2001.

F. H. Hinsley and Alan Stripp (eds.), *Code Breakers: The Inside Story of Bletchley Park*, Oxford University Press, Oxford, 1993.

Andrew Hodges, *Alan Turing: The Enigma*, Vintage, London, 1992.

George Gheverghese Joseph, *The Crest of the Peacock: Non-European Roots of Mathematics*, Penguin Books, London, 1992.

David Knowles, *The Evolution of Medieval Thought*, Vintage Books, New York, 1962.

G. Mander (ed.), `wot txters hav bin w8ing 4`, Michael O'Mara Books, London, 2000.

Leo Marks, *Between Silk and Cyanide: The Story of SOE's Code War*, Harper-Collins, London, 1998.

Michael A. Nielsen and Isaac L. Chuang, *Quantum Information and Quantum Computation*, Cambridge University Press, Cambridge, 2000.

Georges Perec (translated by Gilbert Adair), *A Void*, Harvill Press, 1994.

Edgar Allan Poe, *Complete Tales and Poems*, Castle Books, Edison, NJ, 1985.

Dorothy L. Sayers, *Have His Carcase*, Victor Gollancz, London, 1932.

Simon Singh, *The Code Book: The Secret History of Codes and Code-Breaking*, Fourth Estate, London, 1999.

Douglas R. Stinson, *Cryptography: Theory and Practice* (2nd edition), Chapman & Hall/CRC, Boca Raton, 2002.

Doron Swade, *The Cogwheel Brain: Charles Babbage and the Quest to Build the First Computer*, Little, Brown & Co., London, 2000.

Gordon Welchman, *The Hut Six Story: Breaking the Enigma Codes*, M & M Baldwin, Cleobury Mortimer, 1998.

Ernest Vincent Wright, *Gadsby*, Wetzel Publishing Co., Los Angeles, 1931.

Peter Wright, *Spycatcher: The Candid Autobiography of a Senior Intelligence Officer*, Stoddart Publ. Co., Toronto, 1987.