Coding and Cryptography

Chris Wuthrich

Contents

Essential information for G13CCR

Module : Coding and Cryptography, 10 credits, level 3.

Lecturer : Chris Wuthrich,
`christian.wuthrich@nottingham.ac.uk`,
phone 14920.

Lectures :
- Mondays 15:00 in room C04 in Physics
- Thursdays 10:00 in the same room

Office Hours : In my office C58 in Mathematics on Mondays 12:00 – 15:00. If you wish to meet me at any other time, please contact me by email.

Booklist : See also the official booklist.

- Dominic Welsh, *Codes and cryptography* [21] QA 269.5 WEL
- San Ling and Chaoping Xing, *Coding theory : a first course* [11] QA 268 LIN
- Raymond Hill, *A first course in coding theory* [5] QA 269.6 HIL
- William Stein, *Elementary number theory: primes, congruences, and secrets* available online [16] QA241 STE
- Gareth A. Jones and Mary J. Jones, *Elementary number theory* [6] QA 241 JON
- Henry Beker and Fred Piper, *Cipher systems : the protection of communications* [1] QA 269.5 BEK
- Simon Singh *The code book* [15] Jubilee Library QA 76.9 A25 SIN

Lecture Notes : This booklet. An electronic copy can be found on the moodle webpage `http://moodle.nottingham.ac.uk/course/view.php?id=3660`.

Assessment : The assessment of this module consists of a 2h exam (3 out of 4 questions). Consult the page the feedback page
`http://www.maths.nottingham.ac.uk/MathsModulesFeedback/G13CCR/`
for information on exams in previous years.

Computer software : Not needed at all, but these are the most useful software packages related to this module.

- sage is a free open-source mathematical software which contains a lot of built-in functions for coding and cryptography. See [7] and [13]. You can use it online at `http://www.sagenb.org`.
- pari-gp is a small but very powerful software for doing number theory.

Introduction

These are the lecture notes for the modules G13CCR, Coding and Cryptography, as given in the spring semester 2013 at the University of Nottingham.

Cryptography is the art (or the science) of encrypting messages so that no other than the authorised person can decrypt and read the message. What is coding theory then? It is something quite different.

There are many sorts of code and we will only discuss one type, namely error-correction codes. Other codes are used for compression, but they are (no longer) part of this module. There is a conceptual, mathematical approach called Information Theory, which treats the subjects like entropy and information in transmissions.

This module consists of two parts, one on error-correction codes and one on cryptography.

Coding for Error Detection and Correction

The software for early computer had to be fed to the machine using long paper tape, later came the magnetic tapes. Each character was stored as 7 bits, plus an additional bit, the parity check bit. Simply this last bit was a 1 if within the the 7 bits there were an odd number of 1's. In this way it was possible to check if there was an error. Maybe one bit was not recorded correctly or it was not read correctly. Of course, if two errors appeared in one single block of 7 bits the parity check could not detect the errors. Moreover, even if a single error in the block has been transmitted, there is no way to recover the correct block, as any of the seven bit could have been wrong. This lead Hamming to discover error-correcting codes.

Here is another simple example, the International Standard Book Number ISBN. Any book has a unique code, e.g. the book 'Coding Theory' by San Ling and Chaoping Xing [11] is listed as 0-521-52923-9. The last digit of this code is a check digit. If the previous digits are x_1, x_2, ..., x_9, then the last digit is computed as

$$x_{10} = 1 \cdot x_1 + 2 \cdot x_2 + 3 \cdot x_3 + \cdots + 9 \cdot x_9 \bmod 11\,,$$

where 'mod 11' indicates that we replace the result by the remainder when dividing by 11. An 'X' is used when the remainder is equal to 10. In our example

$$9 \equiv 1 \cdot 0 + 2 \cdot 5 + 3 \cdot 2 + 4 \cdot 1 + 5 \cdot 5 + 6 \cdot 2 + 7 \cdot 9 + 8 \cdot 2 + 9 \cdot 3 \pmod{11}\,.$$

Suppose I made an error when copying the code, e.g. I confused the 1 at the fourth position with a 7. Then the sum above would give

$$0 \equiv 1 \cdot 0 + 2 \cdot 5 + 3 \cdot 2 + 4 \cdot \mathbf{7} + 5 \cdot 5 + 6 \cdot 2 + 7 \cdot 9 + 8 \cdot 2 + 9 \cdot 3 \pmod{11},$$

so we know that there is an error. But the code is even better, suppose I swapped the 2 and 3 in the eighth and ninth position, then again my sum will give a different value as the coefficients are different:

$$8 \equiv 1 \cdot 0 + 2 \cdot 5 + 3 \cdot 2 + 4 \cdot 1 + 5 \cdot 5 + 6 \cdot 2 + 7 \cdot 9 + 8 \cdot \mathbf{3} + 9 \cdot \mathbf{2} \pmod{11}.$$

Again, this code still does not tell us where the error is; also if we made two errors it may well be that the sum equal the check digit.

Error-correcting codes are extremely important in modern telecommunication. When the reception of your mobile is very poor, the antenna does not receive correctly all information your phone has sent. It is then important to be able to recover your message from the partial information that the antenna received.

Another example is data storage. Using magnetic disk as an example, if you were to look at bits coming right off a magnetic disk you would see, on the average, about 1 bit error for every 10^9 bits transferred, i.e. as frequent as every 25 seconds. If you were to look at bits coming right off an optical disk, you would see a much higher rate of errors, about 1 bit error in every 10^5 bits transferred. Without some form of error detection and correction, most storage devices would be too unreliable to be useful. (Source [22])

Cryptography

Cryptography originally deals with the problem of encrypting messages so that nobody but the authorised person can decrypt and read it. It has been used throughout the last 2000 years, but the methods and the problematic have become more and more elaborate. Often codes that were believed to guarantee secrecy were in fact not too difficult to break. Here a text in which Casanova explains how he decrypted a text much to the surprise of Madame d'Urfé.

> J'avais fini par connaître à fond Mme d'Urfé qui me croyais fermement un adepte consommé sous le masque de l'incognito, et cinq ou six semaines après elle se confirma dans cette idée chimérique, lorsqu'elle me demanda si j'avais déchiffré le manuscrit où se trouvait la prétendue explication du *Grand-Œuvre*.
>
> «Oui, lui-dis-je, je l'ai déchiffré et par conséquent lu ; mais je vous le rends en vous donnant ma parole d'honneur que je ne l'ai pas copié, car je n'y ai trouvé rien de nouveau.
>
> — Sans la clef, monsieur, excusez-moi, mais je crois la chose impossible.
>
> — Voulez-vous, madame, que je vous nomme la clef ?
>
> — Je vous en prie. »
>
> Je lui donne la parole, qui n'était d'aucune langue, et voilà ma marquise tout ébahie. «C'est trop, monsieur, c'est trop ! je me croyais seule en possession de ce mot mystérieux, car je le conserve dans ma mémoire, je ne l'ai jamais écrit et je suis certaine de ne l'avoir jamais donné à personne. »

> Je pouvais lui dire que le calcul qui m'avait servi à déchiffrer le manuscrit m'avait naturellement servi à deviner la clef ; mais il me vint la lubie de lui dire qu'un génie me l'avait révélé. Cette sottise me soumit entièrement cette femme vraiment savante, vraiment raisonable. . . sur tout autre point que sur sa marotte. Quoi qu'il en soit, ma fausse confidence me donna sur Mme d'Urfé un ascendant immense : je fus dès cet instant l'arbitre de son âme, et j'ai souvent abusé de mon pouvoir sur elle. Maintenant que je suis revenu des illusions qui ont accompagné ma vie, je ne me le rappelle qu'en rougissant, et j'en fais pénitence par l'obligation que je me suis imposée de dire toute la vérité en écrivant ses mémoires.
>
> La gande chimère de cette bonne marquise était de croire fermement à la possibilité de pouvoir parvenie au colloque avec les génies, avec les esprits qu'on appelle élémentaires. Elle aurait donné tout ce qu'elle possédait pour y parvenir, et elle avait connu des imposteurs qui l'avaient trompée, en la flattant de lui faire atteindre le terme de ses vœux.
>
> «Je ne savais pas, me dit-elle, que votre génie eût le pouvoir de forcer le mien à lui révéler mes secrets.
>
> — Il n'a pas été nécessaire de forcer votre génie, madame, car le mien sait tout de sa propre nature.
>
> — Sait-il aussi ce que je renferme de plus secret dans mon âme ?
>
> — Sans doute, et il est forcé de me le dire si je l'interroge.
>
> — Pouvez-vous linterroger quand vous voulez ?
>
> — Toujours, pourvu qe j'aie du papier et de l'encre. Je puis même le faire interroger par vous en vous disant son nom.
>
> — Et vous me le diriez !
>
> — J'en ai le pouvoir, madame, et pour vous en convaincre, mon génie se nomme Paralis. Faites-lui une question par écrit, comme vous la feriez à un simple mortel : demandez-lui, par exemple, comment j'ai pu déchiffrer votre manuscrit, et vous verrez comme je l'obligerai à vous répondre. »
>
> Mme d'Urfé, tremblante de joie, fait sa question et la met en nombres, puis en pyramide à ma façon, et je lui fais tirer la réponse qu'elle met elle-même en lettres. Elle n'obtint d'abord que des consonnes ; mais moyenant une seconde opération qui donna les voyelles, elle trouva la réponse exprimée en termes fort clairs.
>
> Sa surprise se peignait sur tous ses traits, car elle avait tiré de la pyramide la parole qui était la clef de son manuscrit. Je la quittai, emportant avec moi son âme, son cœur, son esprit et tout ce qui lui restait de bon sens.
>
> Casanova de Seingault, 1725–1789

Probably Madame d'Urfé used a Vigenère cipher, a system for encryption that we will discuss in detail – and also show how it is easily broken, if one has a sufficiently long encrypted text at ones disposal. Until today the race is on between those who wish to encrypt messages and those who try to break codes. Did you know, for instances, that your GSM mobile phone communicates with the antenna using some encryption? But did you also know that this code was broken a few years ago and that it is now possible to find the key and to decrypt your conversation? Of course, there are different levels of security: Whether someone wants to make sure that the CIA can not read a message or whether a subscription television company wants to prevent that the average old lady watches television without paying are two completely different tasks. The book [15] is a very pleasant read on the history of cryptography.

Also there is more to cryptography now than encryption and decryption, modern applications such as secure Internet connections need more, like good signature schemes for instance. Say A wants to send a message to B.

- **Secrecy**: A and B want to make sure that no third party E can read the message.
- **Integrity**: A and B want to make sure to detect when a third party E alters the message.
- **Authenticity**: B wants to be certain that it was A who sent the message.
- **Non-repudiation**: B can prove to a third party that it was A who sent the message.

The last entry is typically very important if you buy an online insurance policy and you want to make a claim, but the insurance company claims you have never signed a policy with them. The authenticity is crucial when you use online banking as you want to be certain that the site you are communicating with is indeed your bank and not some phishing site.

Acknowledgements

These course notes are largely based on Dr. Woodall's previous notes for this course, but I enlarged and modified quite a lot using sources like [11], [4], Tom Körner's course [9], Peter Symonds' course [18] and wikipedia, of course. I would also like to thank Edward Hobbs for many correction of the earlier version.

All computations for this course as well as all plots were made using the computer algebra package `sage` [17], which itself uses `pari-gp` [14]. This is free, open-source software comparable to `maple`, `matlab`, `mathematica` or `magma`. See in particular [7] and [13] for more details how to use `sage` for coding and cryptography.

Chapter I

Error-Correcting Codes

The aim of error-correcting codes is to add redundancy to a message in order to be able correct errors that may occur during the transmission of the message.

I.1 Coding for Noisy Channels

Throughout the whole chapter on error-correction codes will be in the following situation. See also figure I.1.

- A **source** produces a message as a sequence of '0' and '1' only[1].
- In a first step this message is **encoded**. The source message is cut into **sourcewords** $\mathbf{w}$ of the same length k. Then the encoding is a function that associates to each possible sourceword a **codeword** $\mathbf{c}$, again a block of '0' and '1', but of length n. Since two distinct sourcewords will correspond to two distinct codewords (because we want to be able to decode later), we have $n \geqslant k$.
- Then the codewords are sent through the **noisy channel**. Bit after bit, the symbols '0' and '1' are sent through this channel. For each symbol there is a probability p that the noise in the channel corrupts the symbol. Such a channel could be the communication between two antennas, say your mobile phone and the base station antenna. See below for more explanations and examples.
- On the other side of the channel the receiver will receive a word $\mathbf{x}$ of length n, which may or may not be the codeword $\mathbf{c}$. The **error correction** tries to recover the original sent codeword from the received word. If for instance the received word differs only in one symbol (also called **bits**) from a codeword $\mathbf{c}'$, but is not itself a codeword, then it would be a good guess to change it to $\mathbf{c}'$. The hope is that $\mathbf{c}' = \mathbf{c}$.
- In the final step the codeword $\mathbf{c}'$ is **decoded** by finding the corresponding sourceword $\mathbf{w}'$.

[1] In other words, we assume to work only with the *binary* alphabet; it is not hard to generalise to arbitrary alphabets but we will stick to the easiest situation here.

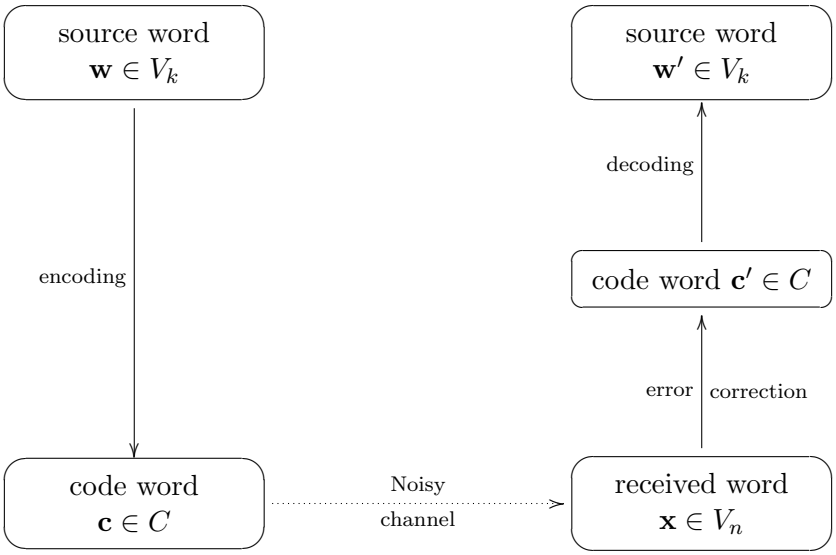


Figure I.1: Coding for a noisy channel

The **code** C is the set of all codewords, usually together with the encoding and decoding function. Let V_n denote the set of all binary words of length n. For example 01001110 is an element of V_8, which we also write as $(0,1,0,0,1,1,1,0)$. We think of elements in V_n as vectors of length n. They will be written in bold face like $\mathbf{x}$ or with a $\vec{\cdot}$ on top, like $\vec{x}$.

As described above, we assume that the set of all sourcewords is V_k for some natural number k. The code is then a well chosen subset C of V_n with a 1-to-1 mapping from V_k to C. Hence

$$\#V_k = 2^k = \#C \leqslant \#V_n = 2^n.$$

Since each codeword will be $n-k$ bits longer, we slow down our communication. The **rate** of the code is defined to be

$$r(C) = \frac{k}{n} \leqslant 1.$$

So a good code should have a large rate.

The second measurement of a good code will be how well we can do the error-correction. The hope is of course that $\mathbf{w}'$ is most often equal to $\mathbf{w}$ or, equivalently, that $\mathbf{c}'$ is equal to $\mathbf{c}$. In other words we need a good procedure to find a codeword "close" to the received word. This is best done with the minimum-distance decoding scheme described in the next section.

Example. As a very simple first example we can imagine the following. We split up the source message '110 001 000 01...' as illustrated in blocks of $k = 3$ bits. As a code we decide to repeat each word thrice; so $n = 3k = 9$ and the first sourceword $\mathbf{w}$ ='110' is encoded to $\mathbf{c}$ ='110110110'. So the code C is the set of all words of length 9 which are composed of three copies of the same word of length 3.

This codeword is then sent through the channel and maybe slightly corrupted by noise in the channel. If the probability p is significantly lower than $\frac{1}{9}$, say $p = 10^{-7}$, we expect only at most one error per transmitted word. So if exactly one error occurs then one of the three copies is different and we choose as a codeword in the error correction step to

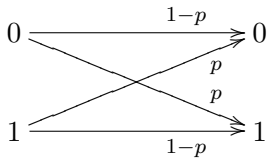
change the one copy that is different. Say we receive $\mathbf{x}$ ='**0**10110110', we will just change the starting 0 back to a 1. Of course, this may go wrong. In the very, very unlikely case that two errors occurred and that they happen to change the same bit, e.g. if we receive $\mathbf{x}$ ='**0**10**0**10110' we would change the last copy instead and then decode to the wrong sourceword $\mathbf{w}'$ ='010'.

This is a very bad code as we will see later. It has some strength for correcting errors, but its rate $\frac{1}{3}$ is very bad. All the transmission is slowed down to a third of the uncoded speed. ◇

A few more words on the channel[2]. The only symbols that can be transmitted are 0 and 1. If $P(a \text{ received } | b \text{ sent})$ denotes the probability that a is received given that b is sent, then

$$P(1 \text{ received } | 0 \text{ sent}) = P(0 \text{ received } | 1 \text{ sent}) = p$$
$$P(0 \text{ received } | 0 \text{ sent}) = P(1 \text{ received } | 1 \text{ sent}) = 1 - p\,.$$

Diagrammatically:



We shall assume[3] that $p < \frac{1}{2}$. Examples of such noisy channels include digital radio transmission, e.g. for mobile phones and Wireless LAN, but also digital transmission through optical fibres and traditional electrical cables. But we can also include storage devices as the channel which starts at the computer writes on a magnetic disk and reads off it later again. As mentioned in the introduction, a Compact Disk has read with probability $p = 10^{-5}$ the wrong bit; while for a magnetic disk the error probability is as low as $p = 10^{-9}$.

The probability that a word of length n is sent through the channel with some corruption is $1-(1-p)^n$, which for small p is approximately $n \cdot p$. The probability that errors occurred at d given places in the word is equal to $p^d \cdot (1-p)^{n-d}$.

Finally, we make an assumption[4] on the source: We suppose that all sourcewords appear with approximately the same frequency and that the probability is not influenced by the word previously sent. This is typically not the case in a language like English as words like 'the' are much more frequent than others. In reality this is not always the case, but it can be achieved by first using a compression algorithm, e.g. "zip" the file. In any case, the restriction is not a serious one.

[2]Such a channel is called a *binary symmetric channel with crossover probability* **p**.
[3]Why? Well, try to think what you would do if $p > \frac{1}{2}$. And what does it mean that $p = \frac{1}{2}$?
[4]This is called a memoryless source.

I.2 The Hamming distance

In this section we will find the best error-correction method. What we hope for is that $\mathbf{c}' = \mathbf{c}$ (referring to the notation in this figure I.1) as then $\mathbf{w} = \mathbf{w}'$. If the received word $\mathbf{w}$ is not a codeword, we wish to interpret it as the codeword *most likely* to have been sent. Intuitively, this is the codeword *closest* to $\mathbf{w}$.

Definition. If $\mathbf{x}, \mathbf{y} \in V_n$, their **Hamming distance** $d(\mathbf{x}, \mathbf{y})$ is the number of positions in which they differ. The **weight** $w(\mathbf{x})$ of $\mathbf{x}$ is the number of 1's in $\mathbf{x}$.

So $w(\mathbf{x}) = d(\mathbf{x}, \mathbf{0})$, where $\mathbf{0} = (0, 0, \ldots, 0)$ is the word with n zeroes. For example, the distance between

$$\begin{array}{rcl} \mathbf{x} & = & (1,\ 0,\ 0,\ 1,\ 1,\ 0) \\ \mathbf{y} & = & (0,\ 0,\ 1,\ 0,\ 1,\ 0) \end{array} \qquad \text{and}$$

is $d(\mathbf{x}, \mathbf{y}) = 3$. The function $d\colon V_n \to \mathbb{N} \cup \{0\}$ satisfies the axioms of a **metric**, which are

- $d(\mathbf{x}, \mathbf{y}) = 0$ if and only if $\mathbf{x} = \mathbf{y}$;
- $d(\mathbf{x}, \mathbf{y}) = d(\mathbf{y}, \mathbf{x})$ and
- $d(\mathbf{x}, \mathbf{y}) + d(\mathbf{y}, \mathbf{z}) \geqslant d(\mathbf{x}, \mathbf{z})$ (**triangle inequality**).

Theorem I.1. *When a word $\mathbf{x} \in V_n$ is received, the codewords $\mathbf{c} \in C$ that are most likely to have been sent are those for which $d(\mathbf{c}, \mathbf{x})$ is smallest.*

Proof. Let $\mathbf{x}$ be a fixed source word. By Bayes's theorem,

$$\begin{aligned} P(\mathbf{c} \text{ sent} \mid \mathbf{x} \text{ received}) &= \frac{P(\mathbf{c} \text{ sent}, \mathbf{x} \text{ received})}{P(\mathbf{x} \text{ received})} \\ &= \frac{P(\mathbf{c} \text{ sent}) \cdot P(\mathbf{x} \text{ received} \mid \mathbf{c} \text{ sent})}{P(\mathbf{x} \text{ received})}. \end{aligned}$$

holds for any code word $\mathbf{c}$. Let

$$h = \frac{P(\mathbf{c} \text{ sent})}{P(\mathbf{x} \text{ received})},$$

which is independent of $\mathbf{c}$, since we are assuming that *all codewords are equally likely*. By the definition of the distance $d = d(\mathbf{c}, \mathbf{x})$, the probability that $\mathbf{x}$ was received, knowing that $\mathbf{c}$ was sent is $p^d \cdot (1-p)^{n-d}$. Thus

$$\begin{aligned} P(\mathbf{c} \text{ sent} \mid \mathbf{x} \text{ received}) &= h \cdot P(\mathbf{x} \text{ received} \mid \mathbf{c} \text{ sent}) \\ &= h \cdot p^d \cdot (1-p)^{n-d} = h \cdot (1-p)^n \cdot \left(\frac{p}{1-p}\right)^d, \end{aligned}$$

which is largest when $d = d(\mathbf{c}, \mathbf{x})$ is smallest, since we are assuming that $p < \frac{1}{2}$ and so $\frac{p}{1-p} < 1$. □

The **ideal observer** always chooses $\mathbf{c}$ so that $P(\mathbf{c}$ sent $\mid \mathbf{w}$ received $)$ is maximal. Theorem I.1 says that, in our setting, the ideal observer uses the **minimum-distance** or **nearest-neighbour** decoding scheme. For complicated codes C it may be very complicated to find the nearest neighbour to a received word. So another scheme may be used in certain cases (see the Reed-Muller codes in section I.11).

Example. This is Hamming's original[5] code. We choose the code C to lie in V_7, while the source word will be in V_4. So this code has rate $\frac{4}{7}$. The code C is defined to be the set of all $\mathbf{c} = (c_1, c_2, \ldots, c_7)$ which satisfy the following parity conditions:

$$\begin{aligned} &c_1 + c_3 + c_5 + c_7 \text{ is even,} \\ &c_2 + c_3 + c_6 + c_7 \text{ is even, and} \\ &c_4 + c_5 + c_6 + c_7 \text{ is even.} \end{aligned} \tag{I.1}$$

For any source word $\mathbf{w} = (w_1, w_2, w_3, w_4)$, we set $c_7 = w_1$, $c_6 = w_2$, $c_5 = w_3$. Now the parity of c_4 is determined, so we set $c_3 = w_4$. The three equation above then determine the parity of c_1, c_2 and c_4, and hence determine completely $\mathbf{c}$. Here is the complete encoding map.

$$\begin{array}{ll} 0000 \mapsto 0000000 & 1000 \mapsto 1101001 \\ 0001 \mapsto 1110000 & 1001 \mapsto 0011001 \\ 0010 \mapsto 1001100 & 1010 \mapsto 0100101 \\ 0011 \mapsto 0111100 & 1011 \mapsto 1010101 \\ 0100 \mapsto 0101010 & 1100 \mapsto 1000011 \\ 0101 \mapsto 1011010 & 1101 \mapsto 0110011 \\ 0110 \mapsto 1100110 & 1110 \mapsto 0001111 \\ 0111 \mapsto 0010110 & 1111 \mapsto 1111111 \end{array} \tag{I.2}$$

Suppose that the code word $\mathbf{c} = 1101001$ is sent. We may or may not receive the correct word $\mathbf{w}$. Here three examples of possible words we could receive:

received word	number of errors	distance to 0000000	1101001	0101010	1000011	...	1111111
1101001	0	4	0	3	3		3
1101000	1	3	1	2	4		4
1101010	2	4	2	1	3		3

Note the number of errors is simply $d(\mathbf{w}, \mathbf{c})$. In the first two cases the code word $\mathbf{c}$ is closest to the received word and so, by choosing the closest neighbour to the received word

[5]Hamming had access to an early electronic computer but was low down in the priority list of users. He should submit his program encoded on paper tape to run over the weekend, but often he would have his tape returned on Monday because the machine had detected an error in the tape. 'If the machine can detect an error' he asked himself 'why can the machine not correct it?' and he came up with the first error-correcting code.

It was easy to implement. It took a little time for his company to realise what he had done, but they were soon trying to patent it. (Source [9])

we correct the error. In the last example, the closest word is 0101010 rather than **c**. So by choosing the code word that was most likely sent, we would actually pick[6] the wrong. See [26] for more pictures and descriptions on this code.

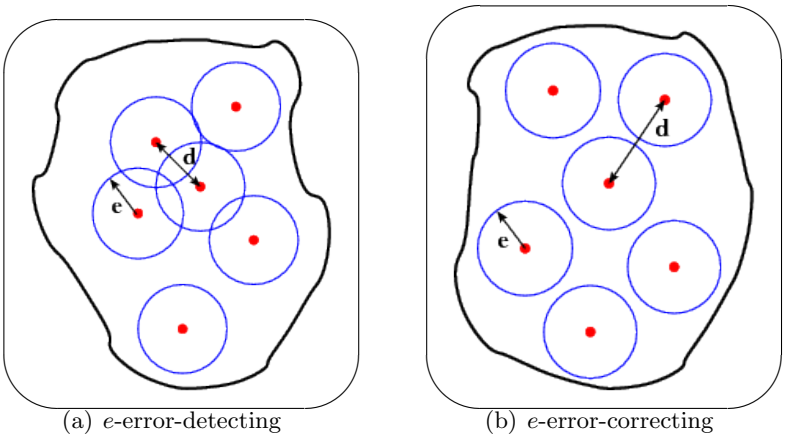
In general it might happen (though not for this code) that there are two or more code words among those closest to the received word, in which case we would have to choose randomly among these and make a mistake with a certain likeliness. ◇

Definition. A code is **e-error-detecting** if it can *detect* that there is a mistake whenever e or fewer errors are made in any one codeword. It is said to be **e-error-correcting** if it can *correct* these errors. The **minimum distance** of a code C is

$$d(C) := \min\{d(\mathbf{c}, \mathbf{c}') \,|\, \mathbf{c} \neq \mathbf{c}' \in C\}\,.$$

It is easy to see that C is e-error-detecting if and only if $d(C) \geqslant e+1$ and that C is e-error-correcting if and only if $d(C) \geqslant 2\,e+1$. So the proof of the following lemma is obvious.

Lemma I.2. *A code C of minimum distance $d(C)$ can detect $d(C)-1$ errors and correct $\lfloor \frac{d(C)-1}{2} \rfloor$ errors, but it cannot do any better than that.*



(a) e-error-detecting (b) e-error-correcting

Figure I.2: Two Potatoes

In an illustrative drawing, we can draw the two pictures in figure I.2. The red dots in the centre of the circles represent the code words in V_n, which is drawn as a potato. The blue circle represent the balls of radius e centred at the code-words. A code is e-error detecting if each ball contains a unique code word. If the circles do not intersect at all, then the code is e error-correcting.

Example. Hamming's code in the previous example has minimum distance $d(C) = 3$. It is therefore 2-error-detecting and 1-error-correcting. ◇

Notation. A **[n, k, d]-code** is a code of length n with 2^k codewords[7] and minimum distance d.

[6] The German word 'verschlimmbessern' describes well how we 'disimproved' the error.

[7] You will also find the notation $(n, 2^k, d)$ instead in the literature.

I.3 Bounds on codes

As seen in figure I.2, it is natural to consider the (closed) **Hamming balls**

$$B(\mathbf{x}, r) = \left\{ \mathbf{y} \in V_n \,\middle|\, d(\mathbf{x}, \mathbf{y}) \leqslant r \right\}.$$

Observe that the balls for varying $\mathbf{x}$ have all the same size; in fact, since all $\mathbf{y}$ of distance j to $\mathbf{x}$ are obtained by switching j coordinates in $\mathbf{x}$, there are exactly $\binom{n}{j}$ such $\mathbf{y}$, and hence

$$\#B(\mathbf{x}, r) = \sum_{j=0}^{r} \binom{n}{j}.$$

It is now easy to prove a first bound on codes.

Theorem I.3 (Sphere packing bound)**.** *If C is a an e-error correcting code in V_n then*

$$\#C \leqslant \frac{2^n}{\sum_{j=0}^{e} \binom{n}{j}}. \tag{I.3}$$

Proof. Each word in V_n appears in at most one of the $\#C$ balls of radius e around the code-words. So $\#C \cdot \#B(\mathbf{x}, e) \leqslant \#V_n$ gives the result. □

Definition. A code is **perfect** if we have an equality in formula (I.3).

In other words, a code is perfect if the balls of radius e around the codewords fill out all of V_n without intersecting. Perfect codes do not exist for all n and e, since the right hand side of (I.3) is not always an integer.

The following codes are further perfect codes.

- The above code of Hamming is perfect.
- Any code with just one codeword.
- The **binary repetition code of length** n, for odd n. Every word in V_n lies within distance $\frac{1}{2}(n-1)$ of exactly one of the two codewords $0\,0\ldots0$ and $1\,1\ldots1$. These are $[n, 1, n]$-codes, usually called **trivial**.
- In section I.10, we will learn about a family of Hamming codes which are all perfect.
- There is a nice 3-error-correcting $[23, 12, 7]$-code, the **binary Golay code**. (See section I.16.6).

Theorem I.4 (The Singleton[8] bound)**.** *Let C be a $[n, k, d]$ code, then $d \leqslant n - k + 1$.*

[8] Named after R. C. Singleton and not the mathematical notion of a set with one single element.

Proof. Take each of the 2^k elements in C and cut off the last $d-1$ bits. Since the distance of C is d, the resulting words of length $n-d+1$ are still distinct, for otherwise we would find two words with distance at most $d-1$ apart from each other. So we have found 2^k distinct elements in V_{n-d+1}, so $k \leqslant n-d+1$. □

Let P_{err} be the average probability that a codeword is incorrectly identified. If no code is used, i.e. if $C = V_k$, then $P_{\text{err}} = 1-(1-p)^k = k \cdot p + \mathbf{O}(p^2)$. A code is good if P_{err} is small, hopefully at least quadratic in p.

Since error-correcting ability is achieved by building redundancy into the message, one might expect that the rate $\frac{k}{n}$ would have to drop down to 0 in order for P_{err} to become smaller and smaller. This is not so.

The **capacity** of the channel with error probability p is defined by the formula

$$C(p) := 1 + p \cdot \log_2(p) + (1-p) \cdot \log_2(1-p).$$

$\log_2$ stands for the logarithm in base 2. Intuitively one should think of it as a measure of how much information can flow through the channel. Shannon developed this notion as part of his Information theory, in which the concepts of information and redundancy are defined rigorously.

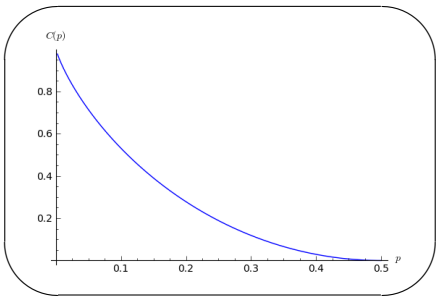


Figure I.3: The capacity

Theorem I.5 (The noisy coding theorem (Shannon 1948)). *Let $\varepsilon > 0$ and $R < C(p)$. There exists a code with rate larger than R and $P_{err} < \varepsilon$.*

But for any code of rate $R > C(p)$ and length n, the error P_{err} is bounded away from 0 by a function in p, R and n which tends to 1 as n grows.

The proof is not given here as it quite long and complicated. If $R < C(p)$, then the existence of a code with small error is proven in a very non-constructive way. In fact the length of this code will be huge. Nobody knows of any practical construction of such codes and, even if they were found, they may well not be efficient to implement at all.

In fact, this theorem plays a central role in coding theory, even though it does not have any useful application, practical or theoretical. It is nevertheless a beautiful and surprising result.

I.4 Some Linear Algebra

Remember that I called V_n a vector space; but over what field? Define the smallest field of all times, denoted[9] by $\mathbb{F}_2$ or $\mathbb{Z}/2\mathbb{Z}$, to be the set $\{0,1\}$ with the multiplication and addition given by

[9] Some people and computer algebra packages such as `sage` [17] will write GF(2) for this field. The worst notation is the most frequently used by computer scientist, namely $\mathbb{Z}_2$ which is easily confused with the ring of so-called dyadic numbers.

$+$	0	1
0	0	1
1	1	0

$\cdot$	0	1
0	0	0
1	0	1

(just as for $0, 1 \in \mathbb{R}$ except that $1 + 1 = 0$). Of course this is the unique field structure on a set of two elements. Note that $a + a = 0$ for all $a \in \mathbb{F}_2$, so addition and subtraction are the same: $a = -a$, and $a + b = a - b$.

Now V_n, the set of binary words of length n, is a n-dimensional vector space over $\mathbb{F}_2$. We will always write elements of V_n as row vectors. Addition in V_n is defined coordinate-wise, like

$$\begin{array}{rcccccc} \mathbf{x} & = & (0, & 0, & 1, & 1, & 0, & 1) \\ +\,\mathbf{y} & = & (0, & 1, & 0, & 1, & 1, & 1) \\ \hline \mathbf{x}+\mathbf{y}=\mathbf{x}-\mathbf{y} & = & (0, & 1, & 1, & 0, & 1, & 0)\,. \end{array}$$

The scalar multiplication is defined in the obvious way

$$0 \cdot \mathbf{x} = \mathbf{0} \qquad \text{and} \qquad 1 \cdot \mathbf{x} = \mathbf{x}$$

with $\mathbf{0}$ being the zero vector $\mathbf{0} = (0, 0, \ldots, 0)$.

Lemma I.6. *For a non-empty set X in V_n it is enough to check that it is closed under addition to assure that it is a linear **subspace**.*

Proof. Suppose X is a non-empty subset of V_n which is closed under addition. In order to prove that X is a linear subspace, we have to make sure that it is also closed under scalar multiplication. Indeed, if $\mathbf{x} \in X$ then $\mathbf{x} + \mathbf{x} = \mathbf{0} \in X$ and hence, for any $\mathbf{x} \in X$ and any $\lambda \in \mathbb{F}_2$, we have $\lambda \cdot \mathbf{x} \in X$. □

As usual, we have the **scalar product**

$$\mathbf{x} \cdot \mathbf{y} = x_1 \cdot y_1 + x_2 \cdot y_2 + \cdots + x_n \cdot y_n$$

with values in $\mathbb{F}_2$; e. g.

$$(1, 0, 1, 1, 0, 1, 1) \cdot (0, 0, 1, 1, 0, 1, 0) = 0 + 0 + 1 + 1 + 0 + 1 + 0 = 1\,.$$

It satisfies the usual properties

$$\mathbf{x} \cdot \mathbf{y} = \mathbf{y} \cdot \mathbf{x}, \quad \mathbf{x} \cdot (\mathbf{y} + \mathbf{z}) = \mathbf{x} \cdot \mathbf{y} + \mathbf{x} \cdot \mathbf{z}, \quad \text{and} \quad (\lambda \mathbf{x}) \cdot \mathbf{y} = \lambda\,(\mathbf{x} \cdot \mathbf{y})\,.$$

Definition. If X is a subset of V_n, we define the **orthogonal complement** by

$$X^{\perp} = \{\mathbf{y} \in V_n \,|\, \mathbf{x} \cdot \mathbf{y} = 0 \text{ for all } \mathbf{x} \in X\}\,.$$

Theorem I.7. *If X is a subspace of V_n of dimension k, then $X^{\perp}$ is a subspace of dimension $n - k$.*

Proof. Clearly $\mathbf{x} \cdot \mathbf{0} = 0$ for all $\mathbf{x}$, so $\mathbf{0} \in X^\perp$. For any $\mathbf{y}$, $\mathbf{z} \in X^\perp$, we have

$$\mathbf{x} \cdot (\mathbf{y} + \mathbf{z}) = \mathbf{x} \cdot \mathbf{y} + \mathbf{x} \cdot \mathbf{z} = 0 + 0 = 0$$

for all $\mathbf{x} \in X$. So $\mathbf{y} + \mathbf{z} \in X^\perp$, hence $X^\perp$ is a subspace.

Now let

$$\begin{aligned} \mathbf{x}_1 &= (x_{11}, x_{12}, \ldots, x_{1n}) \\ \mathbf{x}_2 &= (x_{21}, x_{22}, \ldots, x_{2n}) \\ &\vdots \qquad\quad \vdots \\ \mathbf{x}_k &= (x_{k1}, x_{k2}, \ldots, x_{kn}) \end{aligned}$$

be a basis of X. Then $\mathbf{y} = (y_1, y_2, \ldots, y_n)$ belongs to $X^\perp$ if and only if

$$\begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{k1} & x_{k2} & \cdots & x_{kn} \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}.$$

The coefficient matrix has rank k. Since $X^\perp$ is the solution space to this equation it must be of dimension $n - k$, as required. □

Corollary I.8. $(X^\perp)^\perp = X$.

Proof. Since $\mathbf{x} \cdot \mathbf{y} = 0$ for each $\mathbf{x}$ in X and $\mathbf{y}$ in $X^\perp$, clearly $X \subset X^{\perp\perp}$. But, by the previous theorem I.7,

$$\dim X^{\perp\perp} = n - \dim X^\perp = n - (n - k) = k = \dim X,$$

and so $X^{\perp\perp} = X$. □

I.5 Linear Codes

A **linear code** (of dimension k and length n) is a code C that is a k-dimensional subspace of V_n.

The distance $d(C)$ of a linear code is now simply

$$d(C) = \min\{w(\mathbf{c}) \mid \mathbf{0} \neq \mathbf{c} \in C\} \tag{I.4}$$

since $w(\mathbf{x} - \mathbf{y}) = d(\mathbf{x}, \mathbf{y})$.

Almost all codes used in practice are linear, because:

- encoding and decoding are easy (by matrix multiplication);
- error-correction may be easy, and is certainly relatively straightforward when $n-k$ is small;
- the error-correcting ability of linear codes is easier to determine.

Example. Hamming's original code presented earlier is a linear code. It is a linear $[7,4,3]$-code. The code C lies in V_7, the parity conditions (I.1) for $\mathbf{c}=(c_1,c_2,\ldots,c_7)$ in V_7 to lie in C can be rewritten as a set of linear equation over $\mathbb{F}_2$:

$$\begin{aligned} c_4+c_5+c_6+c_7&=0\\ c_2+c_3+c_6+c_7&=0\\ c_1+c_3+c_5+c_7&=0 \end{aligned} \tag{I.5}$$

This defines a 4-dimensional subspace of V_7. ◇

We introduce some more notations. Given any matrix (or vector), write A^{t} for the transpose matrix (or the corresponding column vector, respectively). In what follows, we will write I_j for the $j\times j$-identity matrix. Also, we will often write block matrices. If, for instance, A is a matrix of size $n\times m$ and B is a matrix of size $k\times m$ then the matrix $C=(A\ B)$ is a matrix of size $(n+k)\times m$.

The following two definitions[10] are crucial for linear codes.

Definition. A matrix G is a **generator matrix** for C if the row vectors of G span the linear subspace C.

In other words, we have that $\mathbf{x}\in C$ if and only if $\mathbf{x}$ is a linear combination of rows of G. Since the only scalars are 0 and 1, we can also say that $\mathbf{x}\in C$ if and only if $\mathbf{x}$ is a sum of some of the rows of G.

Often, we are in the situation where the rows of G form a basis of C. Then there will be exactly k rows. But we might have a generator matrix with more rows in which case they will be linearly dependent, but we will still have that the rank of G is k.

Given a linear code C, we can spell out a generator matrix, by putting elements of C as rows until the rank of the matrix reaches the dimension of C, or equivalently until all remaining elements of C are sums of some of the rows of G.

One can view the generator matrix as a way of giving the **encoding map**

$$\begin{aligned} V_k &\longrightarrow C\\ \mathbf{w} &\longmapsto \mathbf{w}\cdot G \end{aligned}$$

The i^{th} basis vector of V_k is sent to the i^{th} row of G.

Definition. A matrix H is a **parity-check matrix** for C if $C=\ker(H)$, the *kernel* of H.

[10] If you have troubles understanding them, then you should go back to the first year linear algebra. Make sure you fully understand the notions of "basis", "linearly (in)-dependent", "span a subspace", "rank of a matrix", etc. before continuing.

Recall that the kernel of H is the set of all solutions of the system of linear equations $H\mathbf{x}^{\mathrm{t}} = \mathbf{0}^{\mathrm{t}}$. So the parity-check matrix H can be used to give a criterion: A vector $\mathbf{x}$ belongs to C if and only if $H\,\mathbf{x}^{\mathrm{t}} = \mathbf{0}^{\mathrm{t}}$ or, in other words, if and only if $\mathbf{x} \in D^{\perp}$, where D is the subspace generated by the rows of H.

Often, we will be in the situation where the rows of H are linearly independent, in which case there are $n-k$ of them. Both G and H have always n columns.

Example. Giving a parity check matrix for a code is the same as listing linear equations that are satisfied by all elements of C. For the Hamming code given by the equations I.5 it is easy to write out a parity check matrix just by copying these equations.

$$H = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix} \tag{I.6}$$

By solving the equations I.5, we find easily four linearly independent vectors in C. We get a generator matrix by putting them in a matrix as rows.

$$G = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{I.7}$$

In fact, I have taken the image under the encoding map in (I.2) (which happens to be a linear map) of the obvious basis in V_4. So this G corresponds to this encoding map. ◇

Since all rows in G are orthogonal to the rows in H, we have the matrix equation $H \cdot G^{\mathrm{t}} = 0$. Here 0 is the zero-matrix.

Clearly $\operatorname{rank}(G) = \dim C = k$ and $\operatorname{rank}(H) = \dim D = n - k$ by theorem I.7, since $C = D^{\perp}$ and hence, by corollary I.8, $D = D^{\perp\perp} = C^{\perp}$.

Definition. $C^{\perp}$ is the **dual code** of C.

If G and H are a generator matrix and a parity check matrix for C, respectively, then H is a generator matrix for $C^{\perp}$ and G is a parity-check matrix for it.

I.6 The standard form

It is obvious from the previous section that there are several choices of generator and parity check matrices for a linear code C. Here we describe a way of getting them in a standard form.

Definition. Two linear codes are **equivalent** if one can be obtained by applying the same position-wise permutation of bits to all the codewords in the other.

The quality (rate and distance) of two equivalent codes are the same. For all practical purposes, we can exchange a code with an equivalent code without changing the important properties.

Theorem I.9. *By replacing C by an equivalent code if necessary, we can choose*

$$G = \begin{pmatrix} I_k & A \end{pmatrix} \qquad \text{and} \qquad H = \begin{pmatrix} A^t & I_{n-k} \end{pmatrix}$$

for some $k \times (n-k)$ matrix A. The generator matrix G is then said to be in ***standard form****.*

Proof. Let G be a $k \times n$ generator matrix for C. Note that elementary row operations on G do not change C, while permuting the columns corresponds to permuting the bits so as to give a code equivalent to C. Also, we may assume that no row of G is all zero. So permute the columns to ensure $g_{11} = 1$, then add row 1 to other rows as necessary to clear the first column apart from g_{11}. Now permute columns $2, \ldots, n$ to ensure $g_{22} = 1$, then operate with row 2 to clear the second column apart from g_{22}. And so on.[11]

Given $G = \begin{pmatrix} I_k & A \end{pmatrix}$ in standard form, define $H = \begin{pmatrix} A^{\mathrm{t}} & I_{n-k} \end{pmatrix}$. It clearly has rank $n-k$, and so it suffices to show that every row of G is orthogonal to every row of H. But the dot product of the i^{th} row of G with the j^{th} row of H is

$$0 + 0 + \cdots + 0 + a_{ij} + 0 + \cdots + 0 + a_{ij} + 0 + \cdots + 0 = 0.$$

In matrix equations, we can also write

$$H\,G^{\mathrm{t}} = \begin{pmatrix} A^{\mathrm{t}} & I_{n-k} \end{pmatrix} \cdot \begin{pmatrix} I_k \\ A^{\mathrm{t}} \end{pmatrix} = A^{\mathrm{t}} + A^{\mathrm{t}} = 0\,. \qquad \square$$

As an example, we treat again Hamming's code. We start with the matrix G in (I.7). We add the first row to the last two rows.

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

Then add the second row to the first and the third.

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

Now we swap the last two rows.

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

[11] In practice it is preferable to permute rows rather than columns, since this does not change C; but it will not always work.

Finally we add the last row to the second and third row.

$$G' = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix} \tag{I.8}$$

So we have found the matrix A and we can write a new parity check matrix

$$H' = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}. \tag{I.9}$$

Of course, in our operation we have just changed the basis of C, so the rows in G' are still elements in C.

We could have replaced the code by an equivalent code to get to the standard form: Swap the first and last column in G, swap also the second and the sixth and finally move the third column to the forth, the forth to the fifth and the fifth to the third; i.e. apply the permutation $(17)(26)(345)$ to the basis of V_7. We get a generator matrix

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix} \tag{I.10}$$

for a different, but equivalent code.

For instance the encoding map

$$\begin{aligned} V_k &\longrightarrow C \\ \mathbf{w} &\longmapsto \mathbf{w}\, G' \end{aligned}$$

corresponding to G' in (I.8) above sends

$$(w_1, w_2, w_3, w_4) \mapsto (w_1, w_2, w_3, w_4, w_2 + w_3 + w_4, w_1 + w_3 + w_4, w_1 + w_2 + w_4)$$

So clearly the first k coordinates of a code word are just the source word, so we can also easily **decode in standard form**, simply multiply from the right with the matrix

$$\begin{aligned} C &\longrightarrow V_k \\ \mathbf{x} &\longmapsto \mathbf{x} \begin{pmatrix} I_k \\ 0 \end{pmatrix}. \end{aligned}$$

This corresponds to chopping off the $n - k$ last bits. If the generator matrix is not in standard form, then decoding can be quite difficult as it means that we have to solve the inhomogeneous linear equations $\mathbf{x} = \mathbf{w}G$.

I.7 Error correction for linear codes

Definition. The **error-pattern vector** of a received word $\mathbf{x}$ is the vector (word) with 1's where errors have been made. The **corrector** $\mathbf{z}$ is what we add to $\mathbf{x}$ to get a codeword – hopefully the right one.

Recall that according the theorem I.1, the ideal observer is going to choose as the corrector a possible error-pattern vector of minimal weight.

Example. For example, still with Hamming's code, suppose that in the above code C we receive $\mathbf{x} = (1,0,0,0,0,0,1)$. Here a list of some codewords $\mathbf{c}$ and error-pattern $\mathbf{z}$ vectors.

$\mathbf{c}$	$\mathbf{z}$
$(0,0,0,0,0,0,0)$	$(1,0,0,0,0,0,1)$
$(1,0,0,0,0,1,1)$	$(0,0,0,0,0,1,0)$
$(0,1,0,0,1,0,1)$	$(1,1,0,0,1,0,0)$
$(0,0,1,0,1,1,0)$	$(1,0,1,0,1,1,1)$
$(0,0,0,1,1,1,1)$	$(1,0,0,1,1,1,0)$

The error-pattern of minimum weight is $(0,0,0,0,0,1,0)$, so we adopt this as corrector $\mathbf{z}$ and add it to $\mathbf{x}$ to get $(1,0,0,0,0,1,1)$, which is the codeword closest to $\mathbf{x}$ and hopefully the one that was sent. ◇

The set of possible error-pattern vectors is the **coset**

$$\mathbf{x} + C = \left\{ \mathbf{x} + \mathbf{c} \,\middle|\, \mathbf{c} \in C \right\},$$

and a corrector $\mathbf{z}$ is any word of minimum weight in this coset, called the **coset leader**. We can locate the coset by calculating[12] $\mathbf{x} \cdot H^{\mathrm{t}}$, the **error syndrome** or simply **syndrome** of $\mathbf{x}$, since

$$\begin{aligned} \mathbf{x} \cdot H^{\mathrm{t}} = \mathbf{y} \cdot H^{\mathrm{t}} \quad &\Longleftrightarrow \quad H \cdot \mathbf{x}^{\mathrm{t}} = H \cdot \mathbf{y}^{\mathrm{t}} \\ &\Longleftrightarrow \quad H \cdot (\mathbf{x}^{\mathrm{t}} - \mathbf{y}^{\mathrm{t}}) = \mathbf{0}^{\mathrm{t}} \\ &\Longleftrightarrow \quad \mathbf{x} - \mathbf{y} \in C \\ &\Longleftrightarrow \quad \mathbf{x} \text{ and } \mathbf{y} \text{ are in the same coset.} \end{aligned}$$

For Hamming's code with the parity check matrix in (I.9), we have

	coset $\mathbf{x}+C$		syndrome $\mathbf{x}H'^{\mathrm{t}}$	coset leader $\mathbf{z}$
$(0,0,0,0,0,0,0)$	$(1,0,0,0,0,1,1)$	$(0,1,0,0,1,0,1)\ \ldots$	$(0,0,0)$	$(0,0,0,0,0,0,0)$
$(0,0,0,0,0,0,1)$	$(1,0,0,0,0,1,0)$	$(0,1,0,0,1,0,0)\ \ldots$	$(0,0,1)$	$(0,0,0,0,0,0,1)$
$(0,0,0,0,0,1,0)$	$(1,0,0,0,0,0,1)$	$(0,1,0,0,1,1,1)\ \ldots$	$(0,1,0)$	$(0,0,0,0,0,1,0)$
$(1,0,0,0,0,0,0)$	$(0,0,0,0,0,1,1)$	$(1,1,0,0,1,0,1)\ \ldots$	$(0,1,1)$	$(1,0,0,0,0,0,0)$
$(0,0,0,0,1,0,0)$	$(1,0,0,0,1,1,1)$	$(0,1,0,0,0,0,1)\ \ldots$	$(1,0,0)$	$(0,0,0,0,1,0,0)$
$(0,1,0,0,0,0,0)$	$(1,1,0,0,0,1,1)$	$(0,0,0,0,1,0,1)\ \ldots$	$(1,0,1)$	$(0,1,0,0,0,0,0)$
$(0,0,1,0,0,0,0)$	$(1,0,1,0,0,1,1)$	$(0,1,1,0,1,0,1)\ \ldots$	$(1,1,0)$	$(0,0,1,0,0,0,0)$
$(0,0,0,1,0,0,0)$	$(1,0,0,1,0,1,1)$	$(0,1,0,1,1,0,1)\ \ldots$	$(1,1,1)$	$(0,0,0,1,0,0,0)$

[12] This is the transposed of $H\mathbf{x}^{\mathrm{t}}$; if we decided in the beginning to write all our vectors vertically instead of horizontally, we would simply apply H on the left to it.

It is surprising here that the non-trivial coset leaders are exactly the words of weight 1. In general we might well have coset leaders of higher weight or we could have that two words of weight 1 belong to the same coset (when $d(C) = 2$)

Note that we do not need to store the cosets. We need only store the table of 2^{n-k} coset leaders indexed by the syndromes, known as the **syndrome look-up table**.

The error-correcting routine is then: for each received word $\mathbf{x}$, calculate the syndrome $\mathbf{x}H^{\mathrm{t}}$, use it to read off the corrector $\mathbf{z}$ from the table, and return the codeword $\mathbf{x}+\mathbf{z} \in C$. (*And then decode it!*)

Example. Here is a second example. Suppose the code is given by the parity check matrix

$$H = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}. \tag{I.11}$$

This is also a code of dimension 4 and length 7, but it is not as good as Hamming's code. We wish to build up the syndrome look-up table. We know that the syndromes are vectors of length 3, the rank of H, which equals the number of its rows here. So we can put all vectors of length in a list. (We usually order them as binary numbers, but that is optional). The result is given below.

Now, we want to find a coset leader for each of the syndromes. Of course, the syndrome 000, will have 0000000 as a coset leader as it corresponds to the code itself. All other entries will have weight at least 1. We start by looking at 1000000. Its syndrome equals the first column of H, so we can put this vector 1000000 as the coset leader for 110. Next the vector 0100000 is a coset leader for 100. Then the vector 0010000 has also syndrome 110, but we have filled this space already, and as we need only one coset leader for each syndrome, we can choose which one we want. We keep on filling the coset leaders of weight 1, once we have used them all, we find that the syndrome 101 has not yet been given a coset leader. This is imply because 101 does not appear in H as a column. So we know that no vector of weight 1 belongs to the coset, so we have to try to find a vector of weight 2, i.e. we are looking for two columns in H that sum up to 101. There are several choices now, like the second and the fourth, or the first and the last column. So we can give 0101000 or 1000001 as a coset leader to 101.

syndrome	coset leader
$(0,0,0)$	$(0,0,0,0,0,0,0)$
$(0,0,1)$	$(0,0,0,1,0,0,0)$
$(0,1,0)$	$(0,0,0,0,1,0,0)$
$(0,1,1)$	$(0,0,0,0,0,0,1)$
$(1,0,0)$	$(0,1,0,0,0,0,0)$
$(1,0,1)$	$(0,1,0,1,0,0,0)$
$(1,1,0)$	$(1,0,0,0,0,0,0)$
$(1,1,1)$	$(0,0,0,0,0,1,0)$

Say we receive the word $(1,1,1,1,1,1,1)$. Its syndrome is $(0,1,1)$, so we need to add the corrector $(0,0,0,0,0,0,1)$ to it. The unique (because the coset leader for this syndrome was unique) codeword closest to the received word is $(1,1,1,1,1,1,0)$. ◇

I.8 Minimum distance for linear codes

Proposition I.10. *Let C be a linear code with a parity check matrix H. Then the minimum distance $d(C)$ of the code is the minimum number of columns that add up to zero.*

In particular, the distance is $d \geqslant 2$ if there is no zero column in H. The code is 1-error-correcting, i.e. $d \geqslant 3$, if there is no zero-column and no two equal columns in H. It can now be checked very quickly that the Hamming code has minimum distance $d = 3$. The other code given by the parity check matrix (I.11) instead has minimum distance $d = 2$ as there are two equal columns in H.

Proof. Let $\mathbf{c} \in C$ be of weight a. Then $\mathbf{c} \cdot H^{\mathrm{t}} = \mathbf{0}$; however the product $\mathbf{c} \cdot H^{\mathrm{t}}$ is equal[13] to the sum of the columns where $\mathbf{c}$ has a 1. Hence it is a sum of a columns of H that sum up to $\mathbf{0}$.

For a linear code $d(C)$ is the smallest weight of a non-zero vector in C by (I.4), hence the proposition follows. □

For a linear code we have an exact expression for P_{err}:

Theorem I.11. *Let C be a linear code and fix a syndrome look-up table for C. Let a_i be the number of coset leaders in this table of weight i. Then*

$$P_{err} = 1 - \sum_{i=0}^{n} a_i \, p^i (1-p)^{n-i} \, .$$

Proof. A received word is corrected successfully if and only if the error-pattern vector is one of the coset leaders. The probability that it is a *particular* coset leader $\mathbf{z}$ of weight i is

$$P(\text{error pattern} = \mathbf{z}) = p^i \cdot (1-p)^{n-i},$$

whence the result. □

Note that, if C is e-error-correcting, then $a_i = \binom{n}{i}$ for $0 \leqslant i \leqslant e$, since all words with weight less than e are coset leaders, and so

$$P_{\mathrm{err}} \leqslant 1 - \sum_{i=0}^{e} \binom{n}{i} \cdot p^i \cdot (1-p)^{n-i},$$

which behaves like a constant times p^{e+1} for p close to 0.

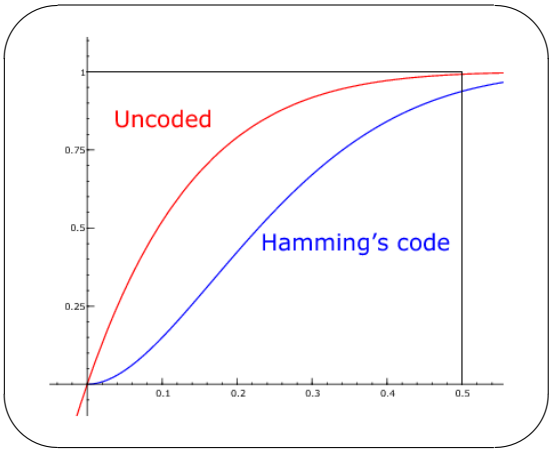


Figure I.4: P_{err} for Hamming's code

[13] For example $(0,1,0,0,0,0,0,1) \cdot H^{\mathrm{t}}$ is the sum of the second and the last column of H.

For Hamming's code, we get

$$\begin{aligned} P_{\text{err}} &= 1 - 1 \cdot (1-p)^7 - 7 \cdot p \cdot (1-p)^6 \\ &= 21\,p^2 + \mathbf{O}(p^3)\,. \end{aligned}$$

If we did not use any code, we would have

$$P_{\text{err}} = 1 - (1-p)^7 = 7\,p + \mathbf{O}(p^2)\,.$$

Hence Hamming's code is vastly better, especially when p is small. See figure I.4.

The code in (I.11) instead has

$$P_{\text{err}} = 1 - 1 \cdot (1-p)^7 - 6 \cdot p \cdot (1-p)^6 - 1 \cdot p^2 \cdot (1-p)^5 = p + 14 \cdot p^2 + \mathbf{O}(p^3)$$

I.9 Linear codes with large distance

We have seen in theorem I.3, that a code had to be small in order to be e-error-correcting. For a linear code of dimension k and length n which is e-error correcting, we have[14]

$$2^{n-k} \geqslant \sum_{j=0}^{e} \binom{n}{j} = \#B(\mathbf{0}, e)\,. \tag{I.12}$$

Our task is to find e-error-correcting codes of length n with high rate, i.e. with large k. Given k and n, the sphere bound is a limit on the largest possible distance d. The next theorem in due to E. N. Gilbert and R. R. Varsharmov and it is a positive result that the distance can be made fairly large.

Theorem I.12. *If*

$$2^{n-k} > \sum_{j=0}^{2e-1} \binom{n-1}{j},$$

then there exists a linear e-error-correcting code C of dimension k and length n.

Proof. By lemma I.2, we want to find a linear $[k, n, 2e+1]$-code. We shall construct an $(n-k)\times n$ matrix H which is the parity check matrix for such a code. From proposition I.10, we see that we want that no sum of $2e$ or fewer columns of H is zero. We construct the columns $\mathbf{x}_1$, $\mathbf{x}_2$,..., $\mathbf{x}_n \in V_{n-k}$ recursively.

First, pick $\mathbf{x}_1$ to be any nonzero word in V_{n-k}. Suppose we have constructed already $\mathbf{x}_1$,..., $\mathbf{x}_r$ for some $r < n$. We wish to choose $\mathbf{x}_{r+1}$ to be any word in V_{n-k} such that $\mathbf{x}_{r+1}$ is not equal to any sum of $2e-1$ or fewer of the previously constructed $\mathbf{x}_i$ with $1 \leqslant i \leqslant r$. That is to say that $\mathbf{x}_{r+1}$ should not be equal to

[14] The inequality can be reinterpreted as follows: The left hand side is the number of distinct syndromes for C. It is easy to see that all elements in the ball $B(\mathbf{0}, e)$ must have distinct syndromes when C is e-error-correcting.

- $\mathbf{0}$ (this excludes 1 element)
- $\mathbf{x}_1$, $\mathbf{x}_2$, ..., $\mathbf{x}_r$ (this excludes r elements)
- $\mathbf{x}_1 + \mathbf{x}_2$, ... (this excludes $\binom{r}{2}$ elements)
- ...
- any sum of $2e-1$ of the $\mathbf{x}_i$ (this excludes at most $\binom{r}{2e-1}$ elements).

So at most

$$1 + r + \binom{r}{2} + \cdots + \binom{r}{2e-1} \tag{I.13}$$

among all $\#V_{n-k} = 2^{n-k}$ are excluded. So if we can prove that the quantity in (I.13) is strictly smaller than 2^{n-k}, then we know that there is at least one possible choice for $\mathbf{x}_{r+1}$. By assumption in the theorem, we have

$$\#V_{n-k} = 2^{n-k} > \sum_{j=0}^{2e-1} \binom{n-1}{j} \geqslant \sum_{j=0}^{2e-1} \binom{r}{j},$$

with the second inequality coming from $r \leqslant n-1$. □

Remark. The above construction proves that such codes exist, but it is not useful in practice. ◇

Both the sphere packing bound and the bound given by the theorem of Gilbert and Varshamov are not optimal. Say, we would like to find 4-error-correcting codes. If $n = 13$, the sphere packing bound gives us that the dimension will be at most 2. Obviously, there is one of dimension 1, but the above theorem does not tell us if there is a 4-error-correcting code with $n = 13$ and $k = 2$. In fact, there is no $[13, 2, 9]$-code.

If instead we ask for $n = 14$, the sphere packing bound tells us again that $k \leqslant 2$ and the above theorem does not help either. But this time, there exists a $[14, 2, 9]$-code.

There are plenty of open problems related to finding better bounds, either lower or upper bounds. For small k and n, the best possible distance are listed in the tables in [3] by Markus Grassl. Clicking on one of the entry will give you a description of the best codes for this n and k.

Here is a table of dimensions and length of 4-error correcting codes. Note that the rate increases.

k	1	2	3	4	5	6	7	8	9	10	11	12	13	14
n	9	14	17	19	20	22	23	25	26	27	29	30	31	32?

It is not known if there exists a 14-dimensional code in V_{32} with minimal distance 9, only one with distance 8 was found so far.

I.10 Hamming Codes

In this section we construct a family of perfect codes. Recall that a code is perfect if we have an equality in (I.12).

For $r \geqslant 2$, let H_r be the $r \times (2^r - 1)$ matrix whose columns are all the nonzero words in V_r, arranged so that the i^{th} column is i in binary. The **binary Hamming code** $\text{Ham}(r)$ is the code of length $2^r - 1$ with H_r as its parity-check matrix (not in standard form). For example if $r = 2$,

$$H_2 = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix}$$

so $\text{Ham}(2)$ is the 1-dimensional code generated by $(1, 1, 1)$, i.e. it is the trivial $[3, 1, 3]$-code. For $r = 3$, we get

$$H_3 = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

Surprise, surprise, this is the ever recurring example which was called Hamming's original code.

Let $1 \leqslant j \leqslant 2^r - 1 = n$. Consider $\mathbf{e}_j$ the j^{th} basis element of V_n. By definition $\mathbf{e}_j \cdot H_r^{\text{t}}$ is the vector of length r containing the binary expansion of j. In particular $\mathbf{e}_j \cdot H_r^{\text{t}} \neq \mathbf{0}^{\text{t}}$, and moreover $(\mathbf{e}_j - \mathbf{e}_k) \cdot H_r^{\text{t}} \neq \mathbf{0}^{\text{t}}$ for any $j \neq k$. Therefore the $2^r - 1$ syndromes $\mathbf{e}_j \cdot H_r^{\text{t}}$ are all distinct. Adding $\mathbf{0}$ as a further syndrome, we have all 2^r syndromes. Hence every non-trivial coset has exactly one word of weight 1. Therefore we have shown the following proposition.

Proposition I.13. *The Hamming code* $\text{Ham}(r)$ *for* $r \geqslant 2$ *is a perfect 1-error-correcting code of length* $n = 2^r - 1$ *and dimension* $k = 2^r - r - 1$.

Note that the rate of $\text{Ham}(r)$ is

$$\frac{k}{n} = \frac{2^r - r - 1}{2^r - 1},$$

which tends to 1 as r grows.

I.11 The First Order Reed-Muller Codes

Another important and very much used family of error-correcting codes are the Reed-Muller codes. The NASA used the them for the Mariner and Viking spacecrafts. Let $m \geqslant 1$. The **Reed-Muller code** $\mathbf{R(1, m)}$ is defined by the generator matrix G_m which

is the $(m+1) \times 2^m$ matrix whose i^{th} column is 1 followed by the binary representation of $i-1$. For instance

$$G_1 = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \qquad G_2 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix} \qquad G_3 = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

The code $R(1,m)$ has therefore length $n = 2^m$ and dimension $m+1$. The word $(1,1,\ldots,1)$ is always a codeword as it is the first row in G_m. All other rows have equally many 0's as 1's, since every binary representation appears once as a column. More precisely the i^{th} row from the bottom starts with a block of 2^i zeroes followed by the same number of 1's, then again 2^i zeroes and so on. Hence every codeword apart from $\mathbf{0}$ and $(1,1,\ldots,1)$ has weight 2^{m-1}. So the distance d is 2^{m-1}, therefore $R(1,m)$ is a $[2^m, m+1, 2^{m-1}]$-code. It is $(2^{m-2}-1)$-error-correcting. The rate is $\frac{k}{n} = \frac{m+1}{2^m}$. The big disadvantage for this code is that the syndrome look-up table is huge; it contains 2^{2^m-m-1} entries.

For instance the code $R(1,5)$ was used for spacecrafts where there is quite a lot of noise in the channel. This code is 7-error-correcting with a rate of 0.1875. But the syndrome look-up would contain $2^{26} = 67108864$ entries.

There is a different algorithm for decoding. This is not a minimum distance decoding scheme. We will illustrate the method on the example of $m = 3$. Let $\mathbf{w} = (w_1, w_2, w_3, w_4)$ be a source word and $\mathbf{c} = (c_1, c_2, \ldots, c_8)$ the corresponding codeword. Then we have the following equalities.

$$\begin{aligned} w_4 &= c_1 + c_2 = c_3 + c_4 = c_5 + c_6 \qquad (= c_7 + c_8) \\ w_3 &= c_1 + c_3 = c_2 + c_4 = c_5 + c_7 \qquad (= c_6 + c_8) \\ w_2 &= c_1 + c_5 = c_2 + c_6 = c_3 + c_7 \qquad (= c_4 + c_8) \end{aligned}$$

On receiving a word $\mathbf{x} = (x_1, x_2, \ldots, x_8)$, we try to find a *source* word $\mathbf{y}$ which will hopefully be $\mathbf{w}$. First evaluate $x_1 + x_2$, $x_3 + x_4$, and $x_5 + x_6$, and choose y_4 to be the more frequent in these three values. Similarly we can obtain y_3 and y_2. Finally choose y_1 as the digit that appears more often in

$$\mathbf{x} + y_2 \cdot \mathbf{r}_2 + y_3 \cdot \mathbf{r}_3 + y_4 \cdot \mathbf{r}_4 \,,$$

where $\mathbf{r}_1$, $\mathbf{r}_2$, $\mathbf{r}_3$, and $\mathbf{r}_4$ are the four rows of G_3.

For a general $m \geqslant 3$, the method is similar. One needs to choose y_2, …, y_{m+1} by choosing the more frequent value among $2^{m-1}-1$ values. If we made an error in one of these values y_2, … y_{m+1}, then at least 2^{m-2} errors must have been made in the transmission. If all of them are correctly decoded, then we obtain the wrong value for y_1 only if the transmission contained at least 2^{m-1} errors. Hence this decoding scheme is $(2^{m-2}-1)$-error-correcting, just as the minimum-distance decoding.

Here an example why this decoding scheme does not always yield the nearest codeword. The code $R(1,4)$ is 3-error-correcting. Suppose we received the word

$$\mathbf{x} = (1,0,0,0,1,0,0,0,1,0,0,0,1,1,1,1).$$

The above scheme yields the source word $\mathbf{0}$, but the second row is closer than $\mathbf{0}$ to $\mathbf{x}$:

$$\mathbf{r}_2 = (0,0,0,0,0,0,0,1,1,1,1,1,1,1,1).$$

I.12 Cyclic Codes

Definition. A linear code $C \subset V_n$ is **cyclic** if

$$(c_0, c_1, \ldots, c_{n-1}) \in C \implies (c_{n-1}, c_0, c_1, \ldots, c_{n-2}) \in C\,.$$

We shifted here the index in the vector; we will keep this throughout the section.

Example. For instance the original code of Hamming's given by (I.6) is not cyclic, as one can see best from the list of all code words (I.2). Also the equivalent code (I.2) is not cyclic. But by swapping in this last matrix first the top two rows, then the last two columns and finally the first two columns we get an equivalent code with generator matrix

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

Here the list of all codewords but $\mathbf{0}$ and 1111111.

$$\begin{aligned} \mathbf{c}_1 &= 1000101 & \mathbf{c}_2 &= 0100111 \\ \mathbf{c}_1 + \mathbf{c}_2 &= 1100010 & \mathbf{c}_1 + \mathbf{c}_3 &= 1010011 \\ \mathbf{c}_2 + \mathbf{c}_3 &= 0110001 & \mathbf{c}_1 + \mathbf{c}_2 + \mathbf{c}_4 &= 1101001 \\ \mathbf{c}_1 + \mathbf{c}_3 + \mathbf{c}_4 &= 1011000 & \mathbf{c}_1 + \mathbf{c}_2 + \mathbf{c}_3 &= 1110100 \\ \mathbf{c}_2 + \mathbf{c}_4 &= 0101100 & \mathbf{c}_2 + \mathbf{c}_3 + \mathbf{c}_4 &= 0111010 \\ \mathbf{c}_3 &= 0010110 & \mathbf{c}_3 + \mathbf{c}_4 &= 0011101 \\ \mathbf{c}_4 &= 0001011 & \mathbf{c}_1 + \mathbf{c}_4 &= 1001110 \end{aligned} \tag{I.14}$$

So clearly this is a cyclic code, which is equivalent to Hamming's code. ◇

We identify now the vector V_n with the vector space of **polynomials**[15] in $\mathbb{F}_2[X]$ of degree smaller than n by the following map.

$$(a_0, a_1, \ldots, a_{n-1}) \mapsto a_0 + a_1 \cdot X + \cdots + a_{n-1} \cdot X^{n-1}$$

So from now on, we will not only write elements in V_n as vectors, like $\mathbf{c}$, but also as polynomials $\mathbf{c}(X)$.

Example. The element $\mathbf{c}_1$ in the above code is now also the polynomial

$$\mathbf{c}_1(X) = 1 + X^4 + X^6\,.$$

[15] Polynomials in $\mathbb{F}_2[X]$ are *formal* sums of the form $a_0 + a_1 X + \cdots + a_{n-1}X^{n-1}$ subject to the usual rules of addition and multiplication. It is important to distinguish this polynomial from the map

$$x \mapsto a_0 + a_1 x + \cdots + a_{n-1}x^{n-1}$$

on $\mathbb{F}_2$. For instance the map $x \mapsto x^2 + x$ is the zero-map, but the polynomial $X + X^2$ is not zero.

◇

The above map is clearly linear. We can endow the vector space of these polynomials with a multiplication with the usual polynomial multiplication, but subject to the additional rule that $X^n = 1$. When multiplying two polynomials $\mathbf{f}(X)$ and $\mathbf{g}(X)$, all the terms of degree larger than $n-1$, can be simplified by $X^{n+k} = X^k$. In algebraic terms we identified V_n with the ring

$$\frac{\mathbb{F}_2[X]}{(X^n - 1)\mathbb{F}_2[X]}\,.$$

Example. In V_7, we will have

$$\begin{aligned}\mathbf{c}_1(X) \cdot \mathbf{c}_2(X) &= (1 + X^4 + X^6) \cdot (X + X^4 + X^5 + X^6) \\ &= X + X^4 + X^6 + X^7 + X^8 + X^9 + X^{11} + X^{12} \\ &= X + X^4 + X^6 + 1 + X + X^2 + X^4 + X^5 \\ &= 1 + X^2 + X^5 + X^6\,,\end{aligned}$$

which, by some marvellous miracle, is equal to $\mathbf{c}_1 + \mathbf{c}_3$. ◇

With this multiplication the cyclic permutation in the definition of a cyclic code is simply the multiplication by X as in

$$X \cdot \left(c_0 + c_1 \cdot X + \cdots + c_{n-1} \cdot X^{n-1}\right) = c_{n-1} + c_0 \cdot X + c_1 \cdot X^2 + \cdots + c_{n-2} \cdot X^{n-1}\,.$$

Proposition I.14. *Let C be a cyclic code in V_n, then for all $\mathbf{f}(X) \in V_n$, we have $\mathbf{f}(X) \cdot \mathbf{c}(X) \in C$ for all $\mathbf{c}(X) \in C$.*

Proof. Let $\mathbf{c}(X) \in C$. By the definition of a cyclic code and the above remark, we have $X \cdot \mathbf{c}(X) \in C$. So by induction $X^2 \cdot \mathbf{c}(X)$, $X^3 \cdot \mathbf{c}(X)$, ... all belong to C. Since C is a linear subspace, we have

$$\mathbf{f}(X) \cdot \mathbf{c}(X) = \left(f_0 + f_1 \cdot X + \cdots + f_{n-1} \cdot X^{n-1}\right) \cdot \mathbf{c}(X) \in C\,.$$ □

Definition. A linear subspace I in V_n such that we have $\mathbf{f}(X) \cdot \mathbf{c}(X) \in I$ for all $\mathbf{c}(X) \in I$ and $\mathbf{f} \in V_n$ is called an **ideal**.

Theorem I.15. *The cyclic codes in V_n are exactly the ideals in V_n.*

Proof. We have seen that any cyclic code is an ideal. Conversely, let I be an ideal. It is clear that I is a linear code. It is cyclic because $X \cdot \mathbf{c}(X) \in I$ for all $\mathbf{c}(X) \in I$. □

I.13 Generator polynomial

We will show that to each cyclic code, we can associate a unique polynomial, called the generator polynomial, which characterises the code completely. But first, we need the following fact about polynomials.

Lemma I.16. *Let* $\mathbf{f}(X)$ *and* $\mathbf{g}(X)$ *be two polynomials in* $\mathbb{F}_2[X]$*. Then there exist two polynomials* $\mathbf{q}(X)$ *and* $\mathbf{r}(X)$ *in* $\mathbb{F}_2[X]$ *such that* $\deg(\mathbf{r}(X)) < \deg(\mathbf{g}(X))$ *and*

$$\mathbf{f}(X) = \mathbf{q}(X) \cdot \mathbf{g}(X) + \mathbf{r}(X)\,. \tag{I.15}$$

The polynomial $\mathbf{q}(X)$ *is called the* ***quotient*** *and* $\mathbf{r}(X)$ *the* ***remainder*** *of the division of* $\mathbf{f}(X)$ *by* $\mathbf{g}(X)$*.*

Proof. This is just the usual long division of polynomials. □

Example. If we divide, say $\mathbf{c}_1(X) = 1 + X^4 + X^6$ by $\mathbf{c}_3(X) = X^2 + X^4 + X^5$, we get

$$\begin{array}{r|l}
 & X+1 \\ \hline
X^5+X^4+X^2 & X^6 + X^4 + 1 \\
 & X^6 + X^5 + X^3 \\ \hline
 & X^5 + X^4 + X^3 + 1 \\
 & X^5 + X^4 + X^2 \\ \hline
 & X^3 + X^2 + 1
\end{array}$$

So the quotient is $\mathbf{q}(X) = 1 + X$ and the remainder $\mathbf{r}(X) = 1 + X^2 + X^3$. Or in one equation:

$$1 + X^4 + X^6 = (1 + X) \cdot (X^2 + X^4 + X^5) + (1 + X^2 + X^3)\,.$$

One thing that is really wonderful about $1 + 1 = 0$ is that you can not make any sign errors. ◇

Note that we can replace on both sides of the equality (I.15) the terms of degree higher than $n - 1$ by the rule for V_n and the equation still holds, but with elements in V_n.

Theorem I.17. *Let* C *be a proper*[16] *cyclic code in* V_n*. There exists a unique polynomial* $\mathbf{g}(X) \in V_n$*, called the* ***generator polynomial*** *such that*

$$C = \left\{ \mathbf{f}(X) \cdot \mathbf{g}(X) \,\middle|\, \mathbf{f}(X) \in V_n \right\}.$$

Moreover $\mathbf{g}(X)$ *is the non-zero polynomial of smallest degree in* C *and* $\mathbf{g}(X)$ *divides* $X^n - 1$ *in* $\mathbb{F}_2[X]$*.*

Of course, the divisibility here means that there exists a polynomial $\mathbf{h}(X) \in V_n$ of degree $n - \deg(\mathbf{g}(X))$ such that $\mathbf{g}(X) \cdot \mathbf{h}(X)$ is equal to $X^n - 1$ as polynomials in $\mathbb{F}_2[X]$, or that $\mathbf{g}(X) \cdot \mathbf{h}(X) = 0$ in V_n. This polynomial $\mathbf{h}(X)$ is also unique and it is called the **parity check polynomial** for C.

Example. For example if $n = 3$ and $\mathbf{g}(X) = X + 1$, then $\mathbf{h}(X) = 1 + X + X^2$, since $X^3 - 1 = (X + 1) \cdot (1 + X + X^2)$. ◇

Note that $X^n - 1 = X^n + 1$ as the coefficients are in $\mathbb{F}_2[X]$; but it seems more natural to write the negative sign here, for aesthetic reasons.

[16] Meaning different from V_n and from $\{\mathbf{0}\}$.

The theorem does not apply as such to the full code $C = V_n$ and the trivial code $C = \{\mathbf{0}\}$. We define the generator polynomial of V_n to be $\mathbf{g}(X) = 1$ and the generator polynomial of $\{0\}$ to be $\mathbf{g}(X) = X^n - 1$. Similarly, we define the parity check polynomial of V_n to be $\mathbf{h}(X) = X^n - 1$ and the one of $\{\mathbf{0}\}$ to be $\mathbf{h}(X) = 1$.

Proof. Let $\mathbf{g}(X)$ be a non-zero polynomial in C of smallest degree. If now $\mathbf{c}(X) \in C$, then there are $\mathbf{f}(X)$ and $\mathbf{r}(X)$ with $\mathbf{c}(X) = \mathbf{f}(X) \cdot \mathbf{g}(X) + \mathbf{r}(X)$ and $\deg(\mathbf{r}(X)) < \deg(\mathbf{g}(X))$ by lemma I.16. Since $\mathbf{c}(X) \in C$ and $\mathbf{f}(X) \cdot \mathbf{g}(X) \in C$ by proposition I.14, we also have $\mathbf{r}(X) \in C$. By minimality of $\mathbf{g}(X)$, this implies that $\mathbf{r}(X) = 0$; so all elements in C are of the form $\mathbf{f}(X) \cdot \mathbf{g}(X)$.

Suppose that there are two polynomials $\mathbf{g}_1(X)$ and $\mathbf{g}_2(X)$ of minimal degree in C. Then they both have the same leading term.[17] Hence $\mathbf{g}_1(X) - \mathbf{g}_2(X)$ is an element of C of smaller degree. So $\mathbf{g}_1(X) = \mathbf{g}_2(X)$.

Finally, when dividing $X^n - 1$ by $\mathbf{g}(X)$, we get two polynomials $\mathbf{h}(X)$ and $\mathbf{r}(X)$ such that $X^n - 1 = \mathbf{h}(X) \cdot \mathbf{g}(X) + \mathbf{r}(X)$ as polynomials in $\mathbb{F}_2[X]$ and such that $\deg(\mathbf{r}(X)) < \deg(\mathbf{g}(X)) \leqslant n$. So in V_n this yields $0 = \mathbf{h}(X) \cdot \mathbf{g}(X) + \mathbf{r}(X)$. So $\mathbf{r}(X)$ must belong to C and by minimality of $\mathbf{g}(X)$, it has to be zero. Hence $X^n - 1 = \mathbf{h}(X) \cdot \mathbf{g}(X)$. □

Of course, this is in algebraic terms the proof that V_n is a principal ideal ring (not domain).

Example. Which is the generator polynomial for our favourite code here? Scanning through the list (I.14), we find that

$$\mathbf{g}(X) = \mathbf{c}_1(X) + \mathbf{c}_3(X) + \mathbf{c}_4(X) = 1 + X^2 + X^3$$

Does it divide $X^7 - 1$? Yes, as $X^7 - 1 = (1 + X^2 + X^3) \cdot (1 + X^2 + X^3 + X^4)$. ◇

$$\begin{aligned}
X^2 - 1 &= (1 + X)^2 \\
X^3 - 1 &= (1 + X) \cdot (1 + X + X^2) \\
X^4 - 1 &= (1 + X)^4 \\
X^5 - 1 &= (1 + X) \cdot (1 + X + X^2 + X^3 + X^4) \\
X^6 - 1 &= (1 + X)^2 \cdot (1 + X + X^2)^2 \\
X^7 - 1 &= (1 + X) \cdot (1 + X + X^3) \cdot (1 + X^2 + X^3) \\
X^8 - 1 &= (1 + X)^8 \\
X^9 - 1 &= (1 + X) \cdot (1 + X + X^2) \cdot (1 + X^3 + X^6) \\
X^{10} - 1 &= (1 + X)^2 \cdot (1 + X + X^2 + X^3 + X^4)^2 \\
X^{11} - 1 &= (1 + X) \cdot (1 + X + X^2 + X^3 + X^4 + X^5 + X^6 + X^7 + X^8 + X^9 + X^{10})
\end{aligned}$$

Table I.1: Factorisations of $X^n - 1$

Proposition I.18. *Conversely, each factor $\mathbf{g}(X)$ of $X^n - 1$ gives rise to a cyclic code in V_n.*

[17] Yes, all polynomials in $\mathbb{F}_2[X]$ are monic.

Proof. The set $I = \{\mathbf{f}(X) \cdot \mathbf{g}(X) \,|\, \mathbf{f}(X) \in V_n\}$ is an ideal in V_n. □

Hence we have a classification of all cyclic codes of length n, simply by listing all divisors of $X^n - 1$. In the table I.1, we give a list of the first few factorisations. Note that when n is even, then the polynomials factors into squares of the polynomials involved for $\frac{n}{2}$. Therefore, one restricts usually the attention to odd n, called **separable**[18] cyclic codes.

I.14 Generator and parity check matrices for cyclic codes

Proposition I.19. *Let $\mathbf{g}(X) = g_0 + g_1 X + \cdots + g_{n-k} X^{n-k}$ be the generator polynomial of a cyclic code C of length n. Then C is of dimension k and a generator matrix is given by*

$$G = \begin{pmatrix} g_0 & g_1 & g_2 & \cdots & g_{n-k} & 0 & 0 & \cdots & 0 \\ 0 & g_0 & g_1 & \cdots & g_{n-k-1} & g_{n-k} & 0 & \cdots & 0 \\ 0 & 0 & g_0 & \cdots & g_{n-k-2} & g_{n-k-1} & g_{n-k} & \ddots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & g_0 & g_1 & g_2 & \cdots & g_{n-k} \end{pmatrix}.$$

Proof. We have to show that

$$B = \{\mathbf{g}(X), X\,\mathbf{g}(X), \ldots, X^{k-1}\,\mathbf{g}(X)\}$$

is a basis of the subspace C. They are clearly linearly independent. Let $\mathbf{c}(X) = \mathbf{f}(X) \cdot \mathbf{g}(X)$ be an arbitrary element in C. Recall from theorem I.17 that there is a polynomial $\mathbf{h}(X)$ of degree k such that $\mathbf{g}(X) \cdot \mathbf{h}(X) = 0$ in V_n. Let $\mathbf{r}(X)$ be the remainder and $\mathbf{q}(X)$ the quotient when dividing $\mathbf{f}(X)$ by $\mathbf{h}(X)$. Now

$$\begin{aligned} \mathbf{r}(X) \cdot \mathbf{g}(X) &= \big(\mathbf{f}(X) - \mathbf{q}(X)\,\mathbf{h}(X)\big) \cdot \mathbf{g}(X) \\ &= \mathbf{f}(X) \cdot \mathbf{g}(X) = \mathbf{c}(X) \end{aligned}$$

and since $\mathbf{r}(X)$ has degree smaller than $\deg(\mathbf{h}(X)) = k$, we have shown that $\mathbf{c}(X)$ belongs to the span of the set B. □

Example. So we get yet another generator matrix for the Hamming code

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

◇

[18] Those knowing about Galois theory will know why.

Definition. Given a polynomial $\mathbf{f}(X) = f_0 + f_1\,X + \cdots + f_m\,X^m$ of degree m in $\mathbb{F}_2[X]$, we define the **reverse polynomial** by

$$\breve{\mathbf{f}}(X) = f_m + f_{m-1}\,X + \cdots + f_0\,X^m = X^{\deg(f)} \cdot \mathbf{f}\Big(\frac{1}{X}\Big)\,.$$

For instance the reverse polynomial of $X^n - 1$ is itself.

Lemma I.20. *Let $\mathbf{g}(X)$ and $\mathbf{h}(X)$ be two polynomials in $\mathbb{F}_2[X]$. Put $\mathbf{f}(X) = \mathbf{g}(X)\cdot\mathbf{h}(X)$. Then $\breve{\mathbf{g}}(X)\cdot\breve{\mathbf{h}}(X) = \breve{\mathbf{f}}(X)$.*

Proof. Using the above formula, we get

$$\begin{aligned}\breve{\mathbf{f}}(X) = (\mathbf{g}\cdot\mathbf{h})\breve{}(X) &= X^{\deg(\mathbf{g}\cdot\mathbf{h})}\cdot(\mathbf{g}\cdot\mathbf{h})\Big(\frac{1}{X}\Big)\\ &= X^{\deg(\mathbf{g})+\deg(\mathbf{h})}\cdot\mathbf{g}\Big(\frac{1}{X}\Big)\cdot\mathbf{h}\Big(\frac{1}{X}\Big) = \breve{\mathbf{g}}(X)\cdot\breve{\mathbf{h}}(X).\end{aligned}$$

□

Theorem I.21. *Let C be a cyclic code with parity check polynomial $\mathbf{h}(X)$. Then $\breve{\mathbf{h}}(X)$ is the generator polynomial of the dual code of C. If $\mathbf{h}(X) = h_0 + h_1\,X + \cdots + h_k\,X^k$, then*

$$H = \begin{pmatrix} h_k & h_{k-1} & \dots & h_1 & h_0 & 0 & \dots & 0\\ 0 & h_k & \dots & h_2 & h_1 & h_0 & \ddots & 0\\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots\\ 0 & 0 & 0 & h_k & h_{k-1} & h_{k-2} & \dots & h_0 \end{pmatrix}$$

is a parity check matrix for C.

Proof. By the previous lemma I.20, we have that $\breve{\mathbf{h}}(X)$ divides $X^n - 1$ because $\breve{\mathbf{g}}(X)\cdot\breve{\mathbf{h}}(X) = X^n - 1$.

Let $\mathbf{f}(X)$ be any polynomial of degree smaller than $n-k$. We shall now prove that $\mathbf{f}(X)\cdot\breve{\mathbf{h}}(X)$ belongs to the dual code $C^\perp$. Write $\mathbf{f}(X)\cdot\breve{\mathbf{h}}(X) = a_0 + a_1\,X + \cdots + a_r\,X^r$ for some $r < n$ and $a_r = 1$. Let $\mathbf{c}(X) = c_0 + c_1\,X + \cdots + c_{n-1}\,X^{n-1}$ be any element of C. Since $\mathbf{g}(X)$ is a generator, $\mathbf{c}(X)$ is a multiple of $\mathbf{g}(X)$. But then $\breve{\mathbf{f}}(X)\cdot\mathbf{h}(X)\cdot\mathbf{c}(X)$ is a multiple of $\mathbf{h}(X)\cdot\mathbf{g}(X) = X^n - 1 = 0$ in V_n. So

$$(a_r + a_{r-1}\,X + \cdots + a_0\,X^r)\cdot(c_0 + \cdots + c_{n-1}\,X^{n-1}) = 0$$

in V_n because the first factor is $\breve{\mathbf{f}}(X)\cdot\mathbf{h}(X)$ by the previous lemma. Consider now the coefficient in front of X^r in the above equality. As no simplification of the form $X^{n+i} = X^i$ can interfere in this, we must have $a_0\,c_0 + a_1\,c_1 + \cdots + a_r\,c_r = 0$. In other words the dot-product of the vector $(a_0, a_1, \dots, a_r, 0, \dots, 0)$ and the vector $(c_0, c_1, \dots, c_{n-1})$ is zero. Hence $a_0 + a_1\,X + \cdots + a_r\,X^r$ belongs to $C^\perp$.

Note that by proposition I.19, any element of the cyclic code C' generated by $\breve{\mathbf{h}}(X)$ belongs to $C^\perp$. So C' is a subspace of $C^\perp$. On the one hand, by theorem I.7, the dimension of $C^\perp$ is $n-k$ where $k = \deg\breve{\mathbf{h}}(X)$. On the other hand, by theorem I.17, the dimension of the cyclic code C' is $n-k$. Hence $C' = C^\perp$.

The final statement about the parity check matrix for C, follows from proposition I.19 and the fact that the parity check matrix of C is a generator matrix for $C^\perp$. □

Corollary I.22. *The dual code of a cyclic code is cyclic.*

I.15 Error-correction for cyclic codes

Cyclic codes are linear. So we could just apply the usual method involving the syndrome look-up table to correct. The following theorem gives a much faster method to correct small errors. But note that this will not always correct to the closest codeword as illustrated in the example after the theorem.

Theorem I.23. *Let C be a cyclic code of minimum distance d generated by $\mathbf{g}(X)$. Let $\mathbf{y}(X)$ be a received word in V_n. Let $\mathbf{s}(X)$ be the remainder when dividing $\mathbf{y}(X)$ by $\mathbf{g}(X)$. If the weight of $\mathbf{s}(X)$ is smaller or equal to $\frac{d-1}{2}$, then $\mathbf{s}(X)$ is a corrector for $\mathbf{y}(X)$, that is $\mathbf{y}(X) + \mathbf{s}(X)$ is the nearest codeword to $\mathbf{y}(X)$.*

Proof. When dividing $\mathbf{y}(X)$ by $\mathbf{g}(X)$, we find a quotient $\mathbf{f}(X)$ and a remainder $\mathbf{s}(X)$ with $\mathbf{y}(X) = \mathbf{f}(X) \cdot \mathbf{g}(X) + \mathbf{s}(X)$ and $\deg(\mathbf{s}(X)) < \deg(\mathbf{g}(X))$. So $\mathbf{y}(X) + \mathbf{s}(X) = \mathbf{f}(X) \cdot \mathbf{g}(X)$ is a codeword in C.

Let $\mathbf{c}(X)$ be any other codeword in C. If we assume that $d(\mathbf{s}, \mathbf{0}) \leqslant \frac{d-1}{2}$, then

$$d(\mathbf{c}, \mathbf{y}) \geqslant d(\mathbf{c}, \mathbf{y} + \mathbf{s}) - d(\mathbf{y} + \mathbf{s}, \mathbf{y}) \geqslant d - \frac{d-1}{2} = \frac{d+1}{2} > \frac{d-1}{2}\,.$$

Hence $\mathbf{y}(X) + \mathbf{s}(X)$ is the closest codeword to $\mathbf{y}(X)$. □

In fact one could prove a bit more. Namely that $\mathbf{s}(X)$ is the syndrome of $\mathbf{y}(X)$ with respect to the 'reversed standard form' parity check matrix. See [11] for more details.

If the length n of the code is huge then the computation of the syndrome using the parity check matrix in theorem I.21 will take a long time, even if most of the entries are zero. The computation of the remainder when dividing by $\mathbf{g}(X)$ is fairly fast, especially when the weight of the polynomial $\mathbf{g}(X)$ is small. Nevertheless, to take the remainder $\mathbf{s}(X)$ as a corrector could yield a codeword which is not the closest codeword to the received word $\mathbf{y}(X)$ – and, hence, it is not likely to be the word which was sent.

Example. Let C be the cyclic code in V_5 generated by the polynomial $\mathbf{g}(X) = 1 + X + X^2 + X^3 + X^4$. Suppose we receive the word $\mathbf{y}(X) = 1 + X + X^2 + X^3$. Since the degree of $\mathbf{y}(X)$ is smaller than the degree of $\mathbf{g}(X)$, we immediately know that $\mathbf{s}(X) = \mathbf{y}(X)$ is the remainder when dividing by $\mathbf{g}(X)$. Hence with the above method, we would correct $\mathbf{y}(X)$ to $\mathbf{y}(X) + \mathbf{s}(X) = \mathbf{0}$, while the codeword $\mathbf{g}(X)$ is closer to $\mathbf{y}(X)$ than $\mathbf{0}$. So this method is not even 1-error-correcting.

Of course, this is a binary repetition code of length 5. It has minimal distance 5 and the usual way of correcting is 2-error-correcting. ◇

In table I.2 we give a list of frequently used cyclic codes. They have some cryptic codenames used by those who actually implement them. The code is given by a generating polynomial. Note that not all $\mathbf{g}(X)$ are irreducible polynomials. (Source [24])

Name	$\mathbf{g}(X)$	Use
CRC-1	$X+1$	
CRC-4-ITU	X^4+X+1	telecom. [19]
CRC-5-ITU	$X^5+X^4+X^2+1$	telecom. [19]
CRC-5-USB	X^5+X^2+1	USB token packets
CRC-6-ITU	X^6+X+1	telecom. [19]
CRC-7	X^7+X^3+1	MultiMediaCard
CRC-8-ATM	X^8+X^2+X+1	Asynchronous Transfer Mode (ADSL)
CRC-8-CCITT	$X^8+X^7+X^3+X^2+1$	1-Wire (cheap wireless communication)
CRC-8-D/M	$X^8+X^5+X^4+1$	1-Wire
CRC-8	$X^8+X^7+X^6+X^4+X^2+1$	
CRC-8-SAE J1850	$X^8+X^4+X^3+X^2+1$	
CRC-10	$X^{10}+X^9+X^5+X^4+X+1$	
CRC-12	$X^{12}+X^{11}+X^3+X^2+X+1$	telecommunication
CRC-15-CAN	$X^{15}+X^{14}+X^{10}+X^8+X^7+X^4+X^3+1$	
CRC-16-CCITT	$X^{16}+X^{12}+X^5+1$	Modems, Bluetooth, Point-to-Point Protocol (dial-up internet), Infrared
CRC-16-IBM	$X^{16}+X^{15}+X^2+1$	XMODEM, USB
CRC-24-Radix-64	$X^{24}+X^{23}+X^{18}+X^{17}+X^{14}+X^{11}+X^{10}+X^7+X^6+X^5+X^4+X^3+X+1$	
CRC-32-MPEG2	$X^{32}+X^{26}+X^{23}+X^{22}+X^{16}+X^{12}+X^{11}+X^{10}+X^8+X^7+X^5+X^4+X^2+X+1$	mpeg2
CRC-32-IEEE 802.3	$X^{32}+X^{26}+X^{23}+X^{22}+X^{16}+X^{12}+X^{11}+X^{10}+X^8+X^7+X^5+X^4+X^2+X+1$	ethernet
CRC-32C	$X^{32}+X^{28}+X^{27}+X^{26}+X^{25}+X^{23}+X^{22}+X^{20}+X^{19}+X^{18}+X^{14}+X^{13}+X^{11}+X^{10}+X^9+X^8+X^6+1$	V.42 Modem
CRC-64-ISO	$X^{64}+X^4+X^3+X+1$	Wide Area Networks

Table I.2: Cyclic codes in use

Example. The cyclic code 'CRC-16-CCITT' is generated by $\mathbf{g}(X) = X^{16} + X^{12} + X^5 + 1$. The smallest n such that $\mathbf{g}(X)$ divides $X^n - 1$ is $n = 7 \cdot 31 \cdot 151 = 32767$. Take any message in binary digits with at most $32767 - 16$ bits. Consider it as a polynomial $\mathbf{f}(X)$ in $\mathbb{F}_2[X]$ of degree less than 32751. Then compute $\mathbf{f}(X) \cdot X^{16}$ and the remainder $\mathbf{s}(X)$ thereof when dividing by $\mathbf{g}(X)$. The coded message to send is then $\mathbf{c}(X) = \mathbf{f}(X) \cdot X^{16} + \mathbf{s}(X)$. By construction $\mathbf{c}(X)$ is a multiple of $\mathbf{g}(X)$ and hence it is a codeword. We have only added 16 bits of redundancy.

If we receive a word, any polynomial $\mathbf{y}(X)$ of degree less than 32767. Compute the remainder of $\mathbf{y}(X)$ when dividing by $\mathbf{g}(X)$. If the result is non-zero, we know that $\mathbf{y}(X)$ is not a codeword and we have detected an error. So this method is 1-error-detecting and has a very high rate of $32751/32767 = 0.9995$. It is neither 2-error-detecting nor 1-error-correcting, but it would detect if all the errors in the transmission were made in 15 consecutive bits. (See the discussion of burst errors below). ◇

I.16 Other Topics in Error Correction Codes

I.16.1 Burst Error Correction Codes

In many applications, typical errors do not come alone. It is quite common to have sequences of a certain length of errors, so called **burst of errors**. Many of the implemented cyclic codes have a particularly good behaviour to correct (or detect) burst errors. See chapter 7.5 in [11].

I.16.2 BCH-Codes

These are special cyclic code that are frequently used. The polynomials are constructed using the fact that the multiplicative group of finite fields $\mathbb{F}_{2^d}$ are cyclic groups. In particular the Hamming $[7, 4]$-code is such a BCH-Code. They are named after Hocquenghem, Bose and Ray-Chaudhuri.

I.16.3 Reed-Solomon Codes

These are also linear codes that are much used. But rather than working over the finite field $\mathbb{F}_2$ of two elements they work with larger fields $\mathbb{F}_q$ and their particular structure. They were used for the Voyager missions and they are still in use in every CD-Player.

I.16.4 Quadratic Residue Codes

(Only for G13FNT-people). These codes are special cyclic codes. Let $p \equiv \pm 1 \pmod 8$ be a prime number. Let ζ be a primitive p^{th} root of unity over $\mathbb{F}_2$. Then the cyclic code generated by the polynomial

$$\prod_a (X - \zeta^a),$$

where a runs over all quadratic residues modulo p, is called a quadratic residue code.

For example with $p = 7$, we get Hamming's original code, once again: Any generator of $\mathbb{F}_8^\times$ is a primitive 7^{th} root of unity. For instance the root of $X^3 + X^2 + 1$ is such. But this is nothing else but the polynomial above in this particular case.

I.16.5 Goppa Codes

These codes use algebraic geometry in their construction. Even though they are not always very practical for implementations, they are very important from a theoretical point of view. Among them one finds families of codes that have an asymptotically very good behaviour.

I.16.6 Golay's code

This is a particularly beautiful code, found by Marcel J. E. Golay in 1949, with many links to different subjects in mathematics, like the Leech lattice, the Monstrous moonshine linking modular forms to a huge finite simple group, ...

This is a perfect cyclic code C in V_{23} of dimension 12 of minimum distance 7. In fact it is the unique code with this property. One way of defining it is via the quadratic residue code for $p = 23$. Or, we can simple give a generating polynomial

$$1 + X^2 + X^4 + X^5 + X^6 + X^{10} + X^{11} .$$

Chapter II

Cryptography

First some terminology. **Cryptography** deals with **encryption**, the process of converting ordinary information (**plaintext**) into unintelligible gibberish (**ciphertext**). **Decryption** is the reverse, moving from unintelligible ciphertext to plaintext. A **cipher system** consists of a general method of encipherment, which remains invariant but uses a variable **key**. It is wise to assume that the enemy knows the general method and is missing only the key.

The goal of **cryptanalysis** is to find some weakness or insecurity in a cipher system; usual the main aim is to reproduce the key given a sufficiently large ciphertext.

The first four sections concern classical cryptography. Then after a short interlude about number theory, we will progress to public key cryptography, the more modern approach to cryptography.

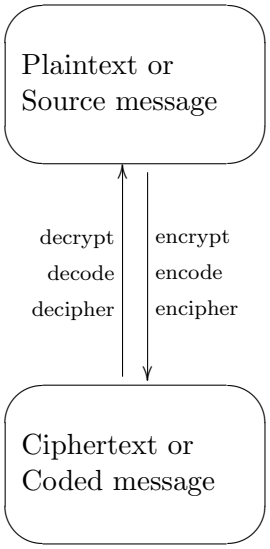


Figure II.1: Cryptography

II.1 Modular Arithmetic

Let $n > 1$ be an integer. We say that a integer a is coprime to n if there is no integer greater than 1 dividing both n and a. For any two integers a and b, we will write $a \equiv b \pmod{n}$ if $a - b$ is divisible by n. The relation $\equiv$ modulo n is an equivalence relation that partitions the integer in **residue classes** of integers having the same remainder when divided by n. The set of all residue classes is denoted by $\mathbb{Z}/n\mathbb{Z}$, which is a set with n elements often represented by $\{0, 1, 2, \ldots, n-2, n-1\}$. The usual addition and multiplication induce operations on $\mathbb{Z}/n\mathbb{Z}$. For instance, we have

$$23 + 34 \equiv 11 + 10 \equiv 21 \equiv 9 \pmod{12} \quad \text{and} \quad 18 \cdot 19 \equiv 6 \cdot 7 \equiv 42 \equiv 6 \pmod{12}.$$

Note however that the division is not always possible. We have

Proposition II.1. *Let $n > 1$ be an integer. Suppose that a is an integer coprime to n. Then there exists a integer b such that $a \cdot b \equiv 1 \pmod{n}$.*

See proposition II.2 for a proof when n is prime and see section II.6.2 for a method to find the integer b. The above proposition is wrong if a is not coprime to n. We call b the **inverse** of a modulo n. We can now divide any residue class x in $\mathbb{Z}/n\mathbb{Z}$ by a simply by multiplying x by b.

For instance if $n = 26$ and $a = 5$ then $b = 21$, since $21 \cdot 5 \equiv 105 \equiv 1 \pmod{26}$. On the other hand, if $a = 12$, then all multiples $a \cdot b$, for any b, are even numbers. Hence even reduced modulo 26, which is also even, we can never be equal to 1. So 12 is not invertible in $\mathbb{Z}/26\mathbb{Z}$.

An affine equation $c \equiv a \cdot x \pmod{n}$ can be solved uniquely in x if a is coprime to n. The solution is $x \equiv b \cdot c \pmod{n}$, where b is the inverse of a modulo n. For instance $17 \equiv 5 \cdot x \pmod{26}$ has as a unique solution $x \equiv 21 \cdot 17 \equiv 19 \pmod{26}$.

If a is not coprime to n, then the equation $c \equiv a \cdot x \pmod{n}$ has either no solution or many solutions. For example $1 \equiv 12 \cdot x \pmod{26}$ has no solution. But $2 \equiv 12 \cdot x \pmod{26}$ has two solution : $x \equiv 11$ or $x \equiv 24 \pmod{26}$. See table II.1 for the list of all inverses modulo 26.

a	1	3	5	7	9	11	15	17	19	21	23	25
b	1	9	21	15	3	19	7	23	11	5	17	25

Table II.1: Inverse modulo 26

A quick note on the fastest way to compute "modulo" on a calculator. Say we wish to compute the remainder of $a = 123456$ modulo $n = 789$. We start by computing a/n: $123456 \div 789 = 156.4714829$ (the number of decimal digits will depend on your display-size). We see that 156 is the integer part of this number, so we subtract it: $156.47142829 - 156 = 0.47142829$. Now we multiply the result with n to get $0.47142829 \times 789 = 372.0000081$. So the answer is 372, i.e. $123456 \equiv 372 \pmod{789}$.

Obviously most computer algebra systems will have the "mod" function implemented. In `pari-gp` [14] one can use `123456 % 789` or `Mod(123456,789)`. The same notations apply to `sage` [17]. In `mathematica` it is `Mod[123456,789]`, in `maple` we write `123456 mod 789` and `mod(123456,789)` in `matlab`.

Please consult [16, chapter 2] for more on modular arithmetic.

II.2 Monoalphabetic Ciphers

Monoalphabetic ciphers are simple permutations of the alphabet, which we will from now on fix as the set of all lower case letters '`a`', '`b`', ... The key is the permutation. We

will present here two special cases of monoalphabetic ciphers, namely the Caesar cipher and the affine cipher.

Caesar cipher

A famous historic example is the **Caesar cipher**, which, according to Suetonius, was used by Julius Caesar to communicate important military information to his generals. Identify the letters in the alphabet with the integers modulo 26 as follows 'a'= 0, ..., 'z'= 25. Encipher by adding κ modulo 26 to each letter, decipher by subtracting κ modulo 26. Julius Caesar used $\kappa = 3$, so the mapping is

plain :	a	b	c	...	w	x	y	z																
cipher:	D	E	F	...	Z	A	B	C																

The key is the value of κ, here 3 or 'D'. There are only 26 possible keys, so one can easily try them all. (This is called the 'brute force attack'.)

Affine cipher

Another example, a little bit more sophisticated is the **affine cipher**. Choose as a key a $\kappa \in {}^{\mathbb{Z}}/_{26\mathbb{Z}}$ and a $\lambda \in {}^{\mathbb{Z}}/_{26\mathbb{Z}}$. But make sure that λ is invertible modulo 26, i.e. that it is the class of an integer which is coprime to 26. Now use the map

$$\begin{aligned} {}^{\mathbb{Z}}/_{26\mathbb{Z}} &\longrightarrow {}^{\mathbb{Z}}/_{26\mathbb{Z}} \\ x &\longmapsto \lambda \cdot x + \kappa \end{aligned}$$

as a permutation of the letters, used for encryption. The decryption is done with the inverse permutation, given by

$$\begin{aligned} {}^{\mathbb{Z}}/_{26\mathbb{Z}} &\longrightarrow {}^{\mathbb{Z}}/_{26\mathbb{Z}} \\ y &\longmapsto \mu \cdot (y - \kappa) = \mu \cdot y + \nu \end{aligned}$$

where μ is the inverse of λ modulo 26 and $\nu \equiv -\mu \cdot \kappa \pmod{26}$. For example with $\lambda = 7$ and $\kappa = 3$, we get the permutation in figure II.2. The decryption map is given by

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
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
3	10	17	24	5	12	19	0	7	14	21	2	9	16	23	4	11	18	25	6	13	20	1	8	15	22
D	K	R	Y	F	M	T	A	H	O	V	C	J	Q	X	E	L	S	Z	G	N	U	B	I	P	W

Figure II.2: An affine encryption table

$y \mapsto 15 \cdot y + 7$.

Despite the fact that this list looks very random, this cipher is not secure at all. In order to break this code, one starts by making a guess for two correspondences. For instance, just from looking at the frequency and positions of letters in a certain given ciphertext,

one might make the guess that e↦Q and t↦X, say. So in the decryption[1] map $y \mapsto \mu \cdot y + \nu$, we must have

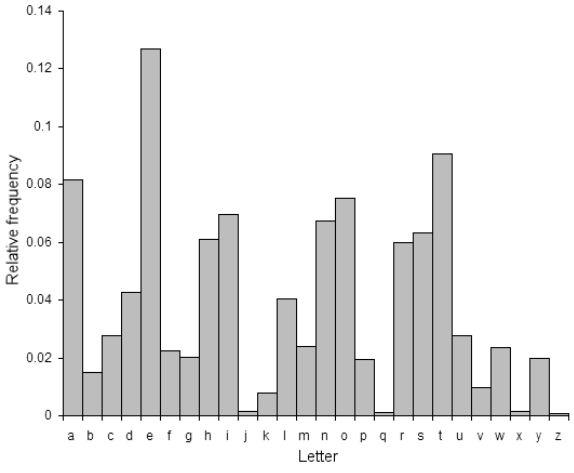
$$\mu \cdot 16 + \nu \equiv 4 \pmod{26}$$
$$\mu \cdot 23 + \nu \equiv 19 \pmod{26}$$

This system of equations modulo 26 has a unique[2] solution $\mu = 17$ and $\nu = 18$. From this is it easy to spell out the full decryption table. Then one checks if the text can be decrypted with this table, if not the guess was wrong and one should try with another guess. Obviously a computer can also use brute force, by testing all $12 \cdot 26$ possible keys.

General attack on monoalphabetic ciphers

It is certainly better to use an arbitrary permutation, rather than these affine maps. Nevertheless these monoalphabetic ciphers are easy to attack with **frequency analysis**.

In figure II.3(a), we see a graphical representation of how often a letter appears in a average English text. In particular, the most frequent letters in the English language are in decreasing order : 'e', 't', 'a', 'o', 'i', 'n', 's', 'h', 'r', 'd', 'l', ... Now we scan the ciphertext to see which are the most frequent letters. Usually one recovers a large part of the original message this way, sufficient to guess the rest. More advanced frequency analysis uses that



(a) Letter frequencies

th	3.21%	the	2.00%
he	3.05%	and	0.93%
in	1.83%	ing	0.74%
er	1.74%	her	0.58%
an	1.73%	tha	0.47%
re	1.37%	hat	0.44%
nd	1.28%	his	0.41%
ed	1.28%	you	0.40%
ha	1.23%	ere	0.39%
es	1.21%	dth	0.35%
ou	1.16%	ent	0.34%
to	1.12%	eth	0.32%
at	1.09%	for	0.32%
en	1.07%	nth	0.31%
on	1.07%	thi	0.30%
ea	1.06%	she	0.30%
nt	1.05%	was	0.29%
st	1.04%	hes	0.29%
hi	1.03%	ith	0.28%

(b) Digram and trigram frequencies

Figure II.3: Frequencies in English (Source [20], see also [2])

[1] One could also determine the encryption map, i.e. solve the equations for λ and κ, but since we are interested in decoding it is often quicker to trying to determine the decryption map directly.

[2] Such systems of equations modulo 26 may also have no solution or two solutions or even 13 solutions. It is important only to take the solutions with μ invertible.

certain couples (digrams) appear more frequently than others. In table II.3(b), you find a list of the most common digrams and trigrams.

II.3 Vigenère Cipher

Another classic encryption is due to Giovan Battista Bellaso in 1553, but it was later attributed to Blaise de Vigenère. It was for long time believe to be unbreakable.

As a key, we fix a word of a certain length k. We add this word repeatedly to the text. As an example suppose the key was 'deadparrot', then we would encode as follows

plain	yeahremarkablebirdthenorvegianbluebeautifulplumageinnit
key	deadparrotdeadparrotdeadparrotdeadparrotdeadparrotdeadp
cipher	BIAKGEDRFDDFLHQIIUHAHROUKEXZOGEPUHQERLHBIYLSAUDRUXLRNLI

(One can do this quickly on `http://www.sharkysoft.com/misc/vigenere/`.) This is a polyalphabetic version of Ceasar's cipher. We could be more general: Take as a key an integer $k > 1$ and permutations π_1, π_2,..., π_k of the alphabet. We encode the first letter using π_1 the second using π_2 and so on.

How can one break this encryption? Note first that, once we know k we can apply frequency analysis to all letters encoded with π_1, then to all letters encoded with π_2 and so on. If the ciphertext is long enough, we will discover the important parts of π_i for all i. In order to find the length k, we can either do it by brute force or with one of the following two ideas.

Babbage's method

Originally[3] the Vigenère code was broken by Charles Babbage, noting that one can spot multiples of the key length from repeating sequences. For instance the word 'the' appears quite often. If the key has length $k = 4$ then roughly every fourth time the word 'the' will be encrypted to the same cipher, say 'skf'. So the sequence 'skf' appears quite often and the distances between these appearances will be a multiple of the length k. Finding several such lengths, the greatest common divisor of them is likely to be the length k.

Friedman's method

Here is a different approach by William F. Friedman, based on coincidence frequencies. Suppose we compare two passages of English text. Suppose, for instance, that the second passage is exactly the same as the first, only shifted by a certain number of places.

```
                                  o  h  th                    e
lookmyladivehadjustaboutenoughofthisthatparrotisdefinitelydeceased
  lookmyladivehadjustabouteno ugh of th isthatparrotisdefinite lydeceased
```

[3]The same method was rediscovered a little bit later by Friedrich Wilhelm Kasiski. This technique is also known as the "Kasiski examination".

```
                                    o  h  th                    e
```

There is a certain number of places where we have by chance the same letter above each other. Here it happened 5 out of 64 times. That is roughly 7.8%. How often do we expect it to happen in average? The probability that a letter is an '`a`' in an English text is about $p_{\text{`a'}} = 0.08167$, so the probability that an '`a`' is on top of an '`a`' is about $p^2_{\text{`a'}} = 0.08167^2$. Hence we find that the probability that two equal letter are above each other is

$$P(\mathrm{id}) = p^2_{\text{`a'}} + p^2_{\text{`b'}} + \cdots + p^2_{\text{`z'}} = \sum_{i=\text{`a'}}^{\text{`z'}} p_i^2 \approx 0.0655\,.$$

Here I used the same the numerical data as in II.3(a). Note that we expect the same probability if we apply the same monoalphabetic cipher to the top and bottom.

Now let π be any permutation of the alphabet. In the example below, I have taken the permutation $\pi(x) = 3\cdot x^5 + 14$ modulo 26, which happens to be a bijection. So $\pi(\text{`a'}) = \text{`O'}$, $\pi(\text{`b'}) = \text{`R'}$, …, $\pi(\text{`z'}) = \text{`L'}$. Apply the permutation to the second line.

```
        a                                                       s
 lookmyladivehadjustaboutenoughofthisthatparrotisdefinitelydeceased
    jeeaywjopmzsvopxiqhoreihsbeiuvedhvmqhvohtoffehmqpsdmbmhsjwpsgsoqgp
        a                                                       s
```

Here we have only 2 cases among the 62 comparisons, i.e. roughly 3.2%. In fact, the average chances to find two equal letters is now

$$P(\pi) = p_{\text{`a'}} \cdot p_{\text{`o'}} + p_{\text{`b'}} \cdot p_{\text{`r'}} + \cdots + p_{\text{`z'}} \cdot p_{\text{`l'}} = \sum_{\text{`a'}}^{\text{`z'}} p_i \cdot p_{\pi(i)}\,,$$

which for our particular π gives 0.0433, clearly less than 0.0655. For an average π, we expect a value of about $\frac{1}{26} \approx 0.0385$. Here the mathematical proof why the permuted version will always have less coincidences: Let π be any non trivial permutation.

$$\begin{aligned} P(\mathrm{id}) - P(\pi) = \sum_i p_i^2 - \sum_i p_i \cdot p_{\pi(i)} &= \frac{1}{2} \cdot \Big(\sum_i p_i^2 + \sum_i p_{\pi(i)}^2 \Big) - \sum_i p_i \cdot p_{\pi(i)} \\ &= \frac{1}{2} \cdot \sum_i \big(p_i - p_{\pi(i)}\big)^2 > 0\,. \end{aligned}$$

Here is how we use it on Vigenère ciphers. Suppose we have a ciphertext C_0 encoded with a Vigenère cipher of an unknown length k and unknown permutations π_1, …, π_k. Let C_m be the text obtained by shifting C_0 by m letters. Define a_m to be the proportion of coincidences that we find when comparing C_0 with C_m.

If m is a multiple of k then the coincidences of the ciphertext are exactly at the same places as coincidences of the m letter shifted plaintexts. So we expect a frequency of about $P(\mathrm{id}) = 6.55\%$. Otherwise, if k is not a multiple of k, then we expect a lower frequency of coincidences: Suppose at the n^{th} place in the plaintext we have the letter α_n. The ciphertext C_0 and C_m agree at the n^{th} position if $\pi_n(\alpha_n) = \pi_{n-m}(\alpha_{n-m})$, i.e. when $\alpha_n = \pi_n^{-1} \circ \pi_{n-m}(\alpha_{n-m})$. So we expect a frequency $P(\pi_n^{-1} \circ \pi_{n-m}) \leqslant P(\mathrm{id})$ with equality only if $\pi_n = \pi_{n-m}$.

We can now quickly compute the values of a_m for various m, and we should have large values when k is a multiple of k. So we can just read off k.

I have encoded the full text of Monty Python's 'Dead Parrot' sketch with a certain key using Vigenère's cipher:

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

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

Now I computed the values of a_m for some m.

m	1	2	3	4	5	6	7	8	9	**10**
a_m	4.617	3.675	3.317	3.720	3.900	4.841	2.689	2.958	4.348	**7.216**
m	11	12	13	14	15	16	17	18	19	**20**
a_m	3.765	3.362	3.631	4.527	4.393	3.810	3.451	4.034	4.617	**6.141**
m	21	22	23	24	25	26	27	28	29	**30**
a_m	3.810	3.675	3.765	3.720	3.407	3.720	3.451	3.317	4.393	**6.186**
m	31	32	33	34	35	36	37	38	39	**40**
a_m	3.989	3.989	3.048	3.586	4.572	4.348	3.451	3.182	3.586	**5.872**

From the fact that the only $a_m > 5$ are exactly the multiples of 10, we should believe that key had length 10. The last step now is to break the ten Caesar ciphers.

II.4 Other Ciphers and Improvements

We saw that we improved the cipher by considering blocks of letters rather than a simple permutation letter by letter. One way of increasing the security is by taking a keyword in the Vigenère cipher that is as long as the plaintext. Typically one would use the page of a book that the sender and the receiver both have. This can again be attacked by a frequency analysis.

Rather than taking a keyword in English, one would better choose a randomly[4] generated string of letters. An example of a perfectly safe cipher system is a **one-time pad**, where the key is a random sequence of letters just as long as the plaintext. This method was used by Soviet spies during the cold war. Though if the key is reused several times, it ceases to be random; then the problem becomes that of the transmission of the key itself.

[4] Actually it is not that easy to generate 'random' text, but we will not discuss this here at all.

Another and better encryption of blocks is to treat them as vectors and to apply a matrix (the key) to it. Rather than working with an alphabet of 26 letters, one usually picks a prime number p, and works with vectors and matrices over the finite field $\mathbb{Z}/p\mathbb{Z} = \mathbb{F}_p$. On the one hand, if the length of the vector is too small, then the security is not sufficient, but, on the other hand, if the size of the matrices are huge, then the multiplication is very time consuming.

All ciphers can be improved, and poor ciphers can be improved greatly, by making the ciphertext more random. Methods include:

- Use abbreviations, acronyms, codenames, unexpected languages, etc., wherever possible.
- Delete spaces and punctuation. Delete the word '`the`' wherever it occurs.
- Replace each space by one of the letters '`v`', '`k`', '`q`', '`j`', '`x`', '`z`' chosen at random.
- Insert two random letters before every plaintext letter before encryption.
- Do the same with the ciphertext after encryption.
- Encode the ciphertext using an e-error-correcting code, but make e random 'errors' in each codeword. This will make no difference to legitimate readers but it will confuse the the illegitimate attacker.

II.5 Block Cipher

Block ciphers are the most used encryption systems. Although not as secure as the public key systems that we will discuss later, they are very fast and provide a sufficient level of security for most applications.

A first block cipher developed in the early 70's, called DES (Data Encryption Standard) that was much used was based on Horst Feistel's idea. It was for a long time the standard system, using a key of length 56 bits. With modern computers this encryption can not be considered secure any more, as it is possible to run through all possible 2^{56} keys[5] (brute force attack). The feasibility of cracking DES quickly was demonstrated in 1998 when a special machine was built by the Electronic Frontier Foundation, a cyberspace civil rights group, at the cost of approximately 250000 US $. This computer decrypted a DES–encrypted message after only 56 hours of work.

Since 2000, the new standard is AES, Advanced Encryption Standard, which operates with keys of length 128, 192 or 256 bits. But DES has not died out, yet, it still lives on in the form of Triple DES, which consists of apply DES three times with three different keys.

[5] It seems that IBM had first planned keys with 128 bits length, but that they were urged by the American National Security Agency to drop down to 56 bits (source [25]).

II.5.1 Feistel Cipher

A **block cipher** enciphers binary plaintext by chopping it up into blocks of a certain length, say 64 bits in the case of DES (and 128 bits in case of AES). This block is split up into two equally long blocks considered as vectors $\mathbf{v}$ and $\mathbf{w}$ over $\mathbb{F}_2$. Then these two vectors are enciphered in a first round using a relatively simple algorithm of the form

$$\mathbf{v}' = \mathbf{w} \quad \text{and} \quad \mathbf{w}' = \mathbf{v} + F(\mathbf{w}, \mathbf{k})$$

where F is a vector-valued function on the block $\mathbf{w}$ and the key $\mathbf{k}$. This procedure is easily reversed by the map

$$\mathbf{w} = \mathbf{v}' \quad \text{and} \quad \mathbf{v} = \mathbf{w}' + F(\mathbf{v}', \mathbf{k})\,.$$

Next we modify the key $\mathbf{k}$ to obtain a new key $\mathbf{k}'$, called subkey.

This round is now repeated with $\mathbf{v}'$, $\mathbf{w}'$ and $\mathbf{k}'$ at the place of $\mathbf{v}$, $\mathbf{w}$ and $\mathbf{k}$. And then again on the result and so on.

II.5.2 DES

Let us explain the block cipher explained above in the example of the Data Encryption Standard DES. The blocks are of length 64 bits and the key $\mathbf{k}$ is of length[6] 56 bits.

Here there are 16 rounds. In each round the function F is computed using four steps

- The vector $\mathbf{w}$ is enlarged from a 32 bit block to a 48 bit block by duplicating certain bits.
- The subkey $\mathbf{k}$ is added to the enlarged vector.
- The result is split up into eight blocks of 6 bits. Each of these blocks is passed through a S-box. This is simply a map S from the set of blocks of 6 bits to the set of blocks of 4 bits. The function S is chosen to be non-linear, in the case of DES it is just given by a large table with all values it.
- Finally the resulting eight blocks of 4 bits are put together to a new vector of 32 bits. At last a permutation of these bits is applied to the vector.

II.5.3 AES

For the new encryption system AES the blocks are of 128, 192 or 256 bits. We explain the ideas for 128 bits. Each block is split up into 16 bytes a_{ij}, arranged in a 4×4 table.

a_{00}	a_{01}	a_{02}	a_{03}
a_{10}	a_{11}	a_{12}	a_{13}
a_{20}	a_{21}	a_{22}	a_{23}
a_{30}	a_{31}	a_{32}	a_{33}

Each byte a_{ij} is considered to be an element of the field $\mathbb{F}_{256}$, the finite field[7] with 256 elements. In each round, we do the following steps.

[6] Actually it is longer but the last bits are just parity check bits.

[7] Don't worry if you do not know what this is. There is a field structure on the set of bytes, but it is not $\mathbb{Z}/256\mathbb{Z}$.

- Apply to each a_{ij} an S-box. Here this is simply given by taking first the inverse element in the field $\mathbb{F}_{256}$. This operation is not $\mathbb{F}_2$-linear. Afterwards an affine transformation is applied.
- The rows are shifted. In fact the first row stays the same, the second row is cyclically rotated by one place to the left, the third row by two and the last row by three places to the left.
- The columns are mixed. Consider each column as a 4-dimensional vector over $\mathbb{F}_{256}$. A certain invertible 4×4-matrix is applied to this vector.
- Finally the subkey is added.

There is a good description of AES in wikipedia [23].

II.6 Number Theory

For the second part of the course, we will need some elementary number theory. The proof in this section are non-examinable, but they may be very helpful for a better understanding of the subject. It is very crucial to understand the computational complexity of the different tasks discussed here. Also you should be able to compute inverses and powers in modular arithmetic.

II.6.1 Fermat's Little Theorem

Proposition II.2. *Let p be an odd prime number and let a be a number coprime to p. Then there is a number b such that $a \cdot b \equiv 1 \pmod{p}$.*

In other words, the set ${}^{\mathbb{Z}}/_{p\mathbb{Z}}$ is a field, usually denoted by $\mathbb{F}_p$.

Proof. Look at the list

$$a, \quad 2\,a, \quad 3\,a, \quad \ldots, \quad (p-1)\,a\,.$$

None of the values can be zero modulo p, since p is a prime number. All of these values are distinct modulo p, because, if $x\,a \equiv y\,a \pmod{p}$, then $(x-y)\cdot a$ is divisible by p and, since a is coprime to p and p is a prime number, the first factor has to be divisible by p, which means that $x = y$ when $1 \leqslant x, y < p$. Hence the above list contains every non-zero residue class modulo p exactly once. In particular, the residue class 1 must appear in the list, say it is $b \cdot a$. □

The next theorem is known as **Fermat's Little Theorem**.

Theorem II.3. *Let p be an odd prime number and let a be an integer coprime to p. Then $a^{p-1} \equiv 1 \pmod{p}$.*

Proof. In the proof of the previous proposition, we have shown that the set of the residue classes of $\{a, 2\,a, \ldots, (p-1)\,\}$ is equal to $\{1, 2, \ldots, (p-1)\}$. Multiply all the elements in this list together

$$a \cdot 2a \cdots (p-1)a \equiv 1 \cdot 2 \cdots (p-1) \pmod{p}\,.$$

which gives

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

Now by the proposition there is an integer b such that $(p-1)! \cdot b \equiv 1 \pmod{p}$, since p can not divide $(p-1)!$ again because p is a prime. After multiplying the last equation with b, we get the formula in the theorem. □

There is also a short combinatorial proof of this theorem.[8]

II.6.2 Euclid's Algorithm

In proposition II.2, we have seen that any number prime to p has an inverse residue class. But how do we compute this? The answer is given by the Euclidean algorithm.

Let n and m to natural numbers. We will suppose that $n > m$. The aim is to compute two integers x and y such that $x \cdot n + y \cdot m = \gcd(n, m)$, where $\gcd(n, m)$ is the largest positive integer that divides both n and m. Use the division with remainder to write

$$n = q_1 \cdot m + r_1$$

where $0 \leqslant r_1 < m$ is the remainder and q_1 is the quotient. Then we repeat the same with n and m replaced by m and r_1 respectively, to get

$$m = q_2 \cdot r_1 + r_2$$

for some q_2 and $0 \leqslant r_2 < r_1$. And then we do the same with r_1 and r_2 and so on. Note that the sequence of remainders r_i is strictly decreasing, so at some step k, we will have $r_k = 0$. Then r_{k-1} is equal[9] to $\gcd(n, m)$. From the line defining r_{k-1}, namely

$$r_{k-3} = q_{k-1} \cdot r_{k-2} + r_{k-1}\,, \tag{II.1}$$

[8] Consider the task to fabricate pearl necklaces with coloured pearls. You want p pearls on a string and you have a different colours at your disposal. How many different non-uni-coloured necklaces can you make? Answer is

$$\frac{a^p - a}{p}\,,$$

since there are a^p choices for the pearls, but a of them are uni-coloured and the p in the denominator appears because by rotating the necklace you will get a different one (since p is prime) that you don't want to count twice. Since this number must be an integer p divides $a^p - a$. Now let b be such that $ab \equiv 1 \pmod{p}$; then

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

[9] There is something to prove here! By the last line, we have $r_{k-2} = q_k \cdot r_{k-1}$; so r_{k-1} divides r_{k-2}. From the previous line (II.1), we now conclude that r_{k-1} also divides r_{k-3} and so on. When we reach the first line, we have shown that r_{k-1} is indeed a common divisor of n and m.

Note by construction, the greatest common divisor d of n and m will divide r_1, too. By induction d will divide all of the remainder. In particular, it will divide r_{k-1}. So $d = r_{k-1}$ is the $\gcd(n, m)$.

we get $\gcd(n, m)$ as a linear combination of r_{k-3} and r_{k-2}. Working our way backwards, we get in the end a linear combination of n and m.

For the above question of finding the inverse of a modulo p, we take $n = p$ and $m = a$, so we get an x and a $y = b$ such that $a \cdot b \equiv x \cdot p + y \cdot a = 1 \pmod{p}$.

Here an example: We try to find the inverse of 22 modulo 31. We find the greatest common divisor by

$$\begin{array}{rcrcl} \boxed{31} & = & 1 \cdot & \boxed{22} & + \; 9 \\ \boxed{22} & = & 2 \cdot & \boxed{9} & + \; 4 \\ \boxed{9} & = & 2 \cdot & \boxed{4} & + \; \mathbf{1} \\ 4 & = & 4 \cdot & 1 & + \; 0 \end{array}$$

No surprise, we found that 31 and 22 are coprime. Now we work our way backward, solving each line on the last term. We find

$$\begin{aligned} 1 &= \boxed{9} - 2 \cdot \boxed{4} \\ &= \boxed{9} - 2\Big(\boxed{22} - 2 \cdot \boxed{9}\Big) = 5 \cdot \boxed{9} - 2 \cdot \boxed{22} \\ &= 5 \cdot \Big(\boxed{31} - \boxed{22}\Big) - 2 \cdot \boxed{22} = 5 \cdot \boxed{31} - 7 \cdot \boxed{22} \end{aligned}$$

So $x = 5$ and $y = -7$. Since $1 \equiv (-7) \cdot 22 \equiv 24 \cdot 22 \pmod{31}$, we have found that the inverse of 22 modulo 31 is 24.

This algorithm is very fast. A clever implementation does not need to run down and up again, but can keep track of the linear combination at each step. It is no problem on a computer to work out inverses of huge integers modulo even huger prime numbers in seconds. For instance computing an inverse modulo the prime

$$\begin{aligned} p &= 2^{160} - 2^{32} - 21389 \\ &= 1461501637330902918203684832716283019651637554291 \end{aligned} \tag{II.2}$$

requires my laptop in average about 9μs.

II.6.3 Fast Exponentiation

Let p be a large number (maybe a prime) and let $0 < a < p$. Suppose we want to compute a^k modulo p for some exponent k. Of course we can first compute the integer a^k and then take its remainder modulo p. Even though multiplication of integers is quite a fast process, this takes absurdly long when the k and a are big. So it is better to first multiply a^2 then take it modulo p, and so on. But we can make it even faster with the following idea, called **fast modular exponentiation**.

Write k in binary expansion

$$k = k_0 + k_1 \cdot 2 + \cdots + k_n \cdot 2^n \,.$$

By definition $k_n = 1$. Start with $b = a$. Now, if k_{n-1} is 1, then we replace b by $a \cdot b^2$ modulo p, otherwise by b^2 modulo p. Then with the same rule for k_{n-2} and so on. In the end b will have the value a^k modulo p. The idea is simply the following equation

$$a^k = a^{k_0} \cdot \left(a^{k_1} \cdot \left(a^{k_2} \left(\cdots \left(a^{k_{n-1}} \cdot (a^{k_n})^2 \right)^2 \cdots \right)^2 \right)^2 \right)^2 .$$

So all we need to do is squaring n times and maybe multiplying a few times by a, always modulo p. For instance suppose we want to compute 3^{41} modulo $p = 101$. As $41 = 2^5 + 2^3 + 1 = 101001_2$, we get

i	5	4	3	2	1	0
k_i	1	0	1	0	0	1
b	3	$3^2 \equiv 9$	$3 \cdot 9^2 \equiv 41$	$41^2 \equiv 65$	$65^2 \equiv 84$	$3 \cdot 84^2 \equiv 59$.

So $3^{41} \equiv 59 \pmod{101}$, which is much better than passing through the computation of $3^{41} = 36472996377170786403$.

Warning: Do not forget that in the first step, we simply copy b; alternatively you can think that you started with $b = 1$ at the very start.

To illustrate how quick this is, my laptop computed $a^k \bmod N$ for randomly chosen integers with 600 decimal digits. In average it took 33 milliseconds. Since this operation is so often used in modern secure communication, it is now no longer just implemented in software, but directly designed in microchips.

II.6.4 Discrete Logarithm

Many public key cipher systems are based on the **discrete logarithm** problem. Let p be a prime; think of a large one. Let $1 < a < p$. Suppose we have a number b which was computed as the power a^k modulo p for some unknown exponent k. How can we find k, given we know a and p ? This number is called the **discrete logarithm** of b in base a modulo p. Of course, we can not use the real logarithms $\log_a$. Note also that the solution will not be unique, since for instance if k is a discrete logarithm, then $k + (p - 1)$ will be a discrete logarithm, too. (Use Fermat's little theorem II.3.)

But unlike for real number where the logarithm can be computed very fast, there is no known fast algorithm to compute this discrete version. If you do find one, you will get very rich and famous. Of course, when p is small, it is quite simple to check the possibilities a, a^2, ... until we hit b. But for large p this may take very, very long.

As we will need a particularly good choice for the basis a, so called primitive elements we delve a bit deeper into number theory here.

Definition. The **multiplicative order** of a modulo p is the smallest $k > 0$ such that $a^k \equiv 1 \pmod{p}$.

It is not difficult to see that $a^k \equiv 1 \pmod{p}$ if and only if k is a multiple of the multiplicative order of a modulo p.

Example. Take $p = 13$ and $a = 3$. Then $a^2 \equiv 9 \pmod{p}$, but $a^3 \equiv 1 \pmod{13}$, so the multiplicative order of 3 modulo 13 is 3. Instead 2 has multiplicative order 12 modulo 13. ◇

The larger the multiplicative order of an element the more distinct powers it will have, since it is easy to see that $a^k \not\equiv a^l \pmod{p}$ if $k \neq l$ are between 0 and the multiplicative order of a. So the larger the multiplicative order, the better a is suited as a basis for the discrete logarithm.

Lemma II.4. *The multiplicative order of a modulo p divides $p-1$.*

Proof. Let r be the multiplicative order of a modulo p. Let n be the smallest integer such that $nr \geqslant p-1$. So $r > nr-(p-1) \geqslant 0$. We have

$$a^{nr-(p-1)} = \left(a^r\right)^n \cdot \left(a^{p-1}\right)^{-1} \equiv 1^n \cdot 1^{-1} = 1 \pmod{p}$$

by Fermat's little theorem II.3 and the definition of r. Now $nr \neq p-1$ would contradict the minimality of r as then $nr-(p-1)$ would be a smaller power of a that is congruent to 1 modulo p. So r divides $p-1$. □

So we see that the largest possible multiplicative order is $p-1$.

Theorem II.5. *Let p be a prime. Then there exists integers $1 < a < p$ of multiplicative order $p-1$. They are called* ***primitive elements*** *or* ***primitive roots*** *modulo p.*

The proof of this theorem is a bit harder, see Chapter 1 in G13FNT or proposition II.1.2. in [8] for more details.

The theorem says that we can always find such a primitive element, but it does not tell us how. For $p = 7$, for instance, we can take $a = 3$, since $3^2 \equiv 2$, $3^3 \equiv 6$, $3^4 \equiv 4$, $3^5 \equiv 5$, and $3^6 \equiv 1$ modulo 7. We could also take $a = 5$, but not $a = 2$, since $2^3 \equiv 1 \pmod{7}$.

In order to check that an integer a is a primitive element modulo p, it suffices to check that $a^d \not\equiv 1 \pmod{p}$ for all prime divisors ℓ of $p-1$ where $d = \frac{p-1}{\ell}$. (This is due to lemma II.4.) There is no general recipe to create a primitive element for a prime p. But there are usually so many that one can be found very quickly by trying randomly some small integers.

Lemma II.6. *If a is a primitive element, then any b coprime to a can be written as a power of a modulo p.*

Proof. The list $1 = a^0, a^1, a^2, \ldots a^{p-2}$ contains only distinct residue classes modulo p. Since there are $p-1$ of them, each invertible one appears exactly once. □

Back to the original question of the discrete logarithm. Let p be a large prime and a a primitive element modulo p. The previous lemma tells us that for any b, there exists a k such that $a^k \equiv b \pmod{p}$. We even know that k is unique modulo $p-1$, but we have no fast method of computing it. Among the best algorithms currently known are the 'baby-step-giant-step', the 'index calculus', and the 'number field sieve'.

For instance, take $p = 10^{20} + 93$. The integer $a = 3$ is a primitive element. Powering a by $k = 82736834255363542111$ requires about 7μs on my laptop. But reversing this, i.e. taking the discrete logarithm of $b = 15622648482551120887$ to base a to recover k takes me more than 3 seconds. If the prime has twice as many digits, the computation will take very, very long. In fact, this does not only depend on the size of p, but also on the size of the largest prime factor of $p-1$, here 507526619771207.

II.6.5 Factoring

Another example of a so-called trap-door function is factorisation. Let p and q be two huge prime numbers. It is very easy to multiply the two number together to get a huge integer $N = p \cdot q$. The reverse operation is very, very difficult. Suppose we are given N, how could we find the factors p and q ? For small N we would just run through the prime numbers up to $\sqrt{N}$ to check if they divide N. But as p and q are large, this will take too much time.

There are now many interesting methods for factoring large integer. Some use so called elliptic curves, other use sieves, but all of them have only limited power to succeed when p and q are big. When one of the primes is of a special form, like when it is too close to a power of 2, or when p and q are too close to each other, then there are methods that work a little bit faster. My favourite algorithm, which in practice factors quite fast, is Pollard's rho, which has also a version used to compute discrete logarithms. It involves two kangaroos jumping around randomly. See [8] for more information on factorisation methods.

Typically on my computer, I can factor

$$4858738916116954251157973608555681449863774770991599 = 571419211300573780561640 71 \cdot 85029323831417276569251769$$

in about 3 seconds. While

$$180859709670463314266764964 6423 \cdot 468208989229517382582817530 4957$$

takes me already 35 seconds. Then

$$111130834478570904995312041180346 6047 \cdot 67082496978155125205424753761964241 03$$

needs more than 4 minutes to factor.

Until a few years ago, the RSA Laboratories [10] maintained a list of large $N = p \cdot q$ with a prize sum for finding its factorisation. The largest N in this list that was factored in 2009 had 230 decimal digits.

$$\begin{aligned} p = {} & 33478071698956898786044169848212690817704794983713768568 91 \\ & 24313889828837938780022876147116525317430877378144679994 89 \\ q = {} & 36746043666799590428244633799627952632279158164343087642 67 \\ & 60322838157396665112792333734171433968102700927987363089 17 \end{aligned} \tag{II.3}$$

Such computations use a very large number of computers working for 3 years. Here it took almost 2000 2.2GHz-Opteron-CPU years to factor it.

One should also add to this discussion that 'quantum computer' would be able to factor integers very fast. Until now such computing methods using quantum physics are only theoretical; but maybe in the future there will be computers that will be able to decipher all the methods discussed here. Then we will have to use other methods, like quantum cryptography.

II.6.6 Primality

In the discussions so far on computational number theory, we often used large prime numbers. So we need a way of testing that a number n is a prime number. At first this problem looks just as difficult as factorisation. But in fact it is much easier.

For instance it is often very easy to prove that a number is composite, even without spotting a factor. Say we want to check if a number n is prime. Choose any integer a. If the $\gcd(a, n)$ is not 1 or n then, we have already found a factor (though that is not likely to happen). So suppose that a and n are coprime. Now compute a^{n-1} modulo n using the fast exponentiation. If the result is not 1, we immediately know that n can not be a prime number by Fermat's Little Theorem II.3. If it is 1, we can still pick another a. Even if we fail thousand times, this method will not prove to us that n is a prime, other, more powerful methods are needed. One of the fastest algorithms uses elliptic curves. Recently a 'deterministic polynomial time' algorithm was found by Manindra Agrawal, Neeraj Kayal, and Nitin Saxena [12]. The latter two were undergraduate students at the time of discovery in 2002.

For instance the huge number in (II.2) was proven to be a prime in 25.1 ms on my computer. Similarly, I can check within 0.1 seconds that the two numbers in (II.3) are indeed primes.

II.7 RSA

In 1977, Ron Rivest, Adi Shamir, and Leonard Adleman found[10] how one could use the factorisation trapdoor to create a cipher system. This method, abbreviated RSA, is among the most used systems nowadays. Most secure internet connections will use RSA. Though because it takes longer to encrypt and decrypt as with classical methods, it is very often only used for an initial key exchange for a classical cipher system, such as DES or AES.

Let us suppose that Alice would like to send a highly secret message to Bob. First, Bob needs to create the keys. He chooses two large[11] prime numbers p and q. Bob then computes $N = p \cdot q$. He then chooses a large $d < (p-1)(q-1)$ which is coprime to[12] $(p-1) \cdot (q-1)$. Using the Euclidean algorithm, he can compute an inverse e of d modulo $(p-1) \cdot (q-1)$. He gives the key (N, e) to Alice. In fact even better, he can publish it to everyone. It is a **public key** cipher system. Meanwhile he keeps p, q and d secret to himself.

Alice can now encrypt the message. She breaks the message into blocks each of which can be represented by an integers m modulo N. She encrypts m into $c = m^e$ modulo N. And she sends the sequence of codetexts c to Bob.

Bob, receiving c, computes $b = c^d$ modulo N using his private key d.

[10] It was revealed much later that Clifford Cocks working for the British Intelligence Agency GCHQ discovered the method earlier. See [15] for more.

[11] Nowadays, many internet applications use 2048 bit encryption, that is p and q have roughly 300 digits.

[12] This is of course $\varphi(N)$ for those who know Euler's φ-function.

Theorem II.7. *The decoded message b is equal to the original plaintext message m.*

Proof. We have $b \equiv c^d \equiv m^{d\,e} \pmod{N}$. From the construction of the keys, we know that

$$d \cdot e = t \cdot (p-1)(q-1) + 1$$

for some integer t. Assume first that m is coprime to p. Then $m^{p-1} \equiv 1 \pmod{p}$ by Fermat's Little Theorem II.3. So we have

$$b \equiv m^{t(q-1)\cdot(p-1)+1} \equiv m \cdot \left(m^{p-1}\right)^{t(q-1)} \equiv m \pmod{p}\,.$$

If p divides m, then p divides b, too, and we find $b \equiv m \pmod{p}$ in all cases. In the same way, b and m are congruent modulo q. Hence $b - m$ is divisible by p and q, so $b \equiv m \pmod{N}$. Since both b and m were taken modulo N, they are smaller than N and hence they agree. □

At present, nobody knows of any way of breaking this code without factoring N. It looks[13] as if the problems are equivalently hard.

The security relies on the computational difficulty to factor integers. The procedure of creating a key and the encryption and decryption algorithm, only use primality testing, Euclid's algorithm and fast modular exponentiation, all of which are considered computationally much easier.

Example. Here is a very simple and naive example. The numbers are chosen so small that you would be able to repeat the calculations by hand; of course for numbers that small it is very easy to break the cipher.

Say Bob chooses the primes $p = 101$ and $q = 103$; so $N = 10403$. He must now choose a secret key d coprime to $(p-1)\cdot(q-1) = 10200$. For instance $d = 19$ will do.

Next he wants to compute the public key. So he computes the inverse of $d = 19$ modulo $(p-1)\cdot(q-1) = 10200$ using Euclid's algorithm. He get $e = 6979$. He now publishes his public key $(N, e) = (10403, 6979)$.

Alice can look this up and use it to send a message. Say the message is $m = 249$ representing the number of British nuclear war heads, a highly sensitive information she should not leak anywhere. She uses fast modular exponentiation to compute the ciphertext $c = m^e \equiv 4679 \pmod{N}$. This is now sent over to Bob.

Bob receiving c computes $b = c^d$ modulo N with his secret key $(N, d) = (10403, 19)$. He gets $4679^{19} \equiv 249 \pmod{N}$. ◇

The security of this cipher does not only rely on the difficulty to factor N. An eavesdropper, usually called Eve, will posses N, e and c. She can also start with her own message m and in this case she will have N, e, c and m and she would like to know d. If she could solve $m \equiv c^d \pmod{N}$ on d she would also find the private key of Bob. But that is a discrete logarithm problem which will be very hard to solve if N and d are sufficiently large.

[13] This is known for a sight modification of RSA, called Rabin's public key system, which, nevertheless, is not much used in practice.

II.8 Elgamal

Taher Elgamal described another public key system based on the discrete logarithm trapdoor.

Suppose Angela would like to send a secret message to Benjamin. To produce a key, Benjamin chooses a large prime p and a primitive element a modulo p. He chooses a private key $1 < d < p-1$. Then he computes $e = a^d$ modulo p. He publishes the key (p, a, e).

Angela, wishing to send the message $1 < m < p$, chooses a random integer $1 \leqslant k < p-1$ and sends to Benjamin the pair (c_1, c_2) with $c_1 \equiv a^k \pmod{p}$ and $c_2 \equiv m \cdot e^k \pmod{p}$.

When receiving (c_1, c_2), Benjamin can simply[14] compute $c_2 \cdot c_1^{-d}$ modulo p. Since

$$m \equiv c_2 \cdot e^{-k} \equiv c_2 \cdot a^{-dk} \equiv c_2 \cdot c_1^{-d} \pmod{p},$$

he has recovered he plaintext message.

Example. To have a concrete example, we disregard again the actual magnitude requested for p. Say $p = 31$ and we encode 'a' by 0, ..., 'z' by 25, the space by 26, the apostrophe by 27, the '.' by 28, the ',' by 29, and finally '?' by 30. We can not take 2 as a primitive element because $2^5 \equiv 1 \pmod{31}$, but $a = 3$ is a primitive element modulo p. Benjamin picks the secret key $d = 17$ and publishes the public key $e = 3^{17} \equiv 22 \pmod{31}$. Angela would like to send

'God made beer because he loves us and wants us to be happy.'[15]

So we start with 'g'= 6, and picks $k = 19$. So she sends $c_1 = a^k = 12$ and $c_2 = m \cdot e^k = 4$, always modulo p. Benjamin can recover $c_2 \cdot c_1^{-d} = 12 \cdot 4^{-17} = 6$. The full ciphertext reads as

(12, 4) (21, 26) (28, 27) (23, 10) (20, 26) (12, 0) (18, 11) (1, 4) (1, 26) (7, 18) (24, 21) (4, 2) (14, 15) (7, 3) (13, 17) (17, 22) (7, 5) (17, 0) (1, 20) (13, 27) (6, 11) (17, 19) (11, 21) (8, 8) (15, 9) (30, 20) (21, 26) (19, 17) (29, 15) (28, 7) (29, 20) (2, 18) (4, 9) (16, 22) (6, 0) (1, 13) (3, 4) (18, 23) (28, 12) (17, 0) (14, 6) (18, 18) (1, 18) (20, 15) (13, 30) (1, 18) (30, 5) (17, 27) (6, 23) (30, 5) (6, 26) (14, 9) (13, 8) (8, 14) (25, 0) (25, 13) (1, 15) (18, 26) (11, 22)

where I have used random k's for each encryption. ◇

Of course, with such a small p as in the above example, it is easy to break the code. There are already couples that repeat themselves in the ciphertext. Instead one should use very large primes.

[14] I should add here how to compute negative powers like c^{-d} modulo p. There are two options. Firstly, one can compute the inverse c^{-1} of c modulo p and then raise it to the d^{th} power. Secondly, one can compute c^{p-1-d} instead, by Fermat's little theorem II.3, this gives the same result and $p-1-d$ is positive if $d < p-1$.

[15] A quote that Benjamin Franklin, one of the Founding Fathers of the United States, never said about beer but more or less about wine, according to wikiquote.

Example. Here is how we should have done it better. Write the message in base 31:

$$\begin{aligned} m &= 6 + 14 \cdot 31 + 3 \cdot 31^2 + \cdots + 28 \cdot 31^{58} \\ &= 908289982288042737370816910283164225433124808479820231126415862903363383411423328504027 9 \end{aligned}$$

Then choose a prime of this size, in fact $p = m + 94$ is the next[16] larger prime. The integer 2 is a primitive element. Benjamin creates the e by picking

$$d = 347840360339172928472098974252373088580408407687610717678795628174284850261469697356595 2$$

and raising 2 to the d^{th} power. Angela chooses

$$k = 317057417075497108522112483849644277717028975622864199914784670123145468739408802244198 4$$

and can then encrypt m to

$$\begin{aligned} c_1 &= 350928712078211256885403008970556554081020626821866715717006243239747731506534783732246 8 , \\ c_2 &= 381246359528445944610470608936598679698748057015613688450886843103573357098792736453770 3 . \end{aligned}$$

Now if Benjamin evaluates $c_2 \cdot c_1^{-d}$ he will recover the plaintext m. Writing it in base 31, he will recover the text. ◇

If someone can compute the discrete logarithm of the public key e in base a modulo p, then he would be able to decrypt as he is in the possession of the secret key d. There does not seem any way of decrypting other than solving the discrete logarithm problem.

This system is also widely used. A variant of this idea is used for signatures as we will see later in II.11.2.

Note that this cipher is vulnerable to the "homomorphism attack". Say I want to send $m = 123456789$, representing the amount of Swiss Francs in my secret bank account. So I sent a certain (c_1, c_2). Now someone – it must be that Eve again – intercepts the transmission. Although she can not read it, she can still alter it before sending it forward. Say she replaces c_2 by $1000 \cdot c_2$. Now the receiver, will compute $(1000 \cdot c_2) \cdot c_1^{-1} \equiv 1000 \cdot m = 123456789000$. By the way, RSA has the same problem.

II.9 Diffie-Hellmann Key Exchange

In the sixties, when the banks started to use electronic means to communicate with their branches all over the world, they used classical symmetrical cipher systems. But in order to guarantee a certain level of security the keys needed to be changed often. Of course, it would not be safe to send them electronically, so they had to be distributed by hand.

Whit Diffie and Martin E. Hellman discovered[17] in 1976 a way to exchange keys electronically without having the fear that an eavesdropper would be able to deduce the key.

[16] Of course, this not a good choice, it should be an arbitrary prime.

[17] Again it seems that the British GCHQ was earlier, Malcolm J. Williamson had discovered it before, but it was kept classified.

As in the Elgamal cipher, it is based on the discrete logarithm. A common prime p, large as usual, and a primitive root a are fixed and known to everybody. Each participant of the communication, say Александра and Борис, choose secretly an integer d_A and d_B, respectively. They both send their public key $q_A = a^{d_A}$ and $q_B = a^{d_B}$ to the other. Борис in Moscow can compute $k = a^{d_A \cdot d_B}$ by taking the d_B-th power of Александра's public key q_A, while Александра in her Dacha can compute k by raising Борис' public key q_B to d_A-th power. So without having to travel, they were able to compute each the same number k, which can now be used as a key in a classical symmetric cipher system. An eavesdropper, probably called Ивлина this time, would not be able to compute k, as all she knows is p, a, p_A and p_B. If she can solve the discrete logarithm problem on either Александра's or Борис' public key, she would break the code.

For instance, Александра and Борис could fix $p = 101$ and $a = 2$. She picks the secret key $d_A = 7$, while he chooses $d_B = 21$ secretly. She sends him $q_A = 2^7 \equiv 27 \pmod{101}$ and receives from him $q_B = 2^{21} \equiv 89 \pmod{101}$. Both of them can now compute the key $k = 27^{21} \equiv 89^7 \equiv 63 \pmod{101}$.

Nevertheless there is a big problem with this key exchange system. It is called the Man-In-the-Middle attack. Suppose the evil Ивлина is not only listening to the communication between Александра and Борис, but could also alter it. She then intercepts the communication, chooses her own private key d_E, she sends to Борис her public key $p_E = a^{d_E}$, pretending to be Александра. Борис will agree on a key with Ивлина and all the ciphertext can be read by Ивлина. On the other side she does the same with Александра, pretending to be Борис. Not only can she read all of the communication, she can even alter it without that either Александра not Борис will ever notice it.

In order to protect the communication from such attacks it is important that it is used together with signature and authentication schemes to which we turn our attention later.

II.10 No-Key Protocol

This relatively simple encryption system was proposed by Shamir. A version of it is called Massey-Omura cryptosystem, but I do not know if either is used in real life.

Suppose Ali G would like to send a box containing some important documents to Borat. He closes the box and locks it with a key and sends it to Borat. He can not open it, of course, but he can add a second lock to the box and send it back to Ali. Ali can now remove his lock which he knows how to open and send the box once more to Borat without having to worry about security as there is still Borat's lock on the box. Borat can now open the box.

The fascinating thing about this system is that we never had to communicate a key to the other party. Based on the discrete logarithm problem, we can propose the following cryptosystem.

Annebäbi and Barthli agree as usual on p and a and choose each their secret key d_A and d_B. Annebäbi who wants to tell Barthli some important message m, sends first m^{d_A} to Barthli. He can not read the message, but he can compute it to the power of d_B and so

sending back to Annebäbi the ciphertext $m^{d_A\,d_B}$. She knows how to compute an inverse e_A modulo $p-1$ to d_A. She then sends back to him the message received from Barthli raised to the e_A-th power. There exists an integer k such that $d_A \cdot e_A = k(p-1)+1$, so

$$\left(m^{d_A\,d_B}\right)^{e_A} = m^{d_A\,d_B\,e_A} = \left(\left(m^{p-1}\right)^k \cdot m\right)^{d_B} \equiv m^{d_B} \pmod{p}\,.$$

Barthli can raise this to the e_B-th power, where e_B is an inverse of d_B modulo $p-1$, to recover the plaintext m by the same argument as above.

This system is also vulnerable to the Man-in-the-Middle attack. It does not need an exchange of keys, but it needs twice as much transmission than a system based on an exchange of keys.

II.11 Signatures

II.11.1 Authentication

Amélie receives a message, ostensibly from Bartholomé. How can she confirm that he really sent it? Using any public-key cryptosystem, she can encrypt some arbitrary item using Batholomé's public key and ask him to send back the decrypted version. Only Bartholomé can do this, using his private key.

There is still the problem that Amélie needs to be sure that the public key she found in a public directory is really created by Bartholomé. Companies, banks or credit card provider use the services of a Trusted Third Party, such as VeriSign.

II.11.2 Digital Signature Algorithm

This is a standard algorithm that is used a lot when using secure internet connections. It is similar to Elgamal's signature algorithm, but a little bit faster and therefore more popular.

Assume Aurora wants to add a signature s to her message m she wants to send to Bacchus. We will discuss later what is a good signature. Let suppose now that s is some large integer. As in the Elgamal encryption scheme, we suppose that Aurora and Bacchus have agreed on a prime p and a primitive element a. Aurora has announced publicly her key $q_A = a^{d_A} \bmod p$.

To sign her message, she does now the following. She chooses a random integer k, which should be coprime to $p-1$. First she evaluates $r = a^k \bmod p$. Next she computes an inverse l to k modulo $p-1$. (Not p !) Then she sends to Bacchus the triple (s, r, t) where

$$t = l \cdot (s + d_A\, r) \mod (p-1)\,.$$

Note that she has used here her private key, so nobody else could produce this t. But before sending she checks whether t is coprime to $p-1$; if not she has to do it over again with another k.

Now, Bacchus wants to verify the signature. He can compute an inverse u to t modulo $p-1$. Then he put $v = s \cdot u$ and $w = r \cdot u$, still modulo $p-1$. Now he evaluates $a^v \cdot q_A^w \bmod p$ and checks if it agrees with r modulo p.

Note that this really produces r if the signature is correct as

$$a^v \cdot q_A^w \equiv a^{v+d_A \cdot w} \equiv a^{u \cdot (s+d_A \cdot r)} \equiv a^k \equiv r \pmod{p},$$

since

$$k \equiv k \cdot u \cdot t \equiv k \cdot u \cdot l \cdot (s + d_A \cdot r) \equiv u \cdot (s + d_A \cdot r) \pmod{p-1}.$$

Here again the primes chosen for internet communications are at least 160 bits, but to be on the sure side current recommendations for U.S governmental offices are up to 3072 bits, that is almost 1000 decimal digits.

II.11.3 Hash Functions

A **hash function** takes as an input a text of any length and returns a short string of letters, but in such a way that, given an output it is computationally difficult to construct a second different input with the same output. Of course, this is not really a mathematical definition. These hash function are used to produce signatures.

Suppose Angelina wishes to put a signature to her message m, which could be an electronic contract for instance. She computes the value of $s = H(m)$ where H is one of the standard hash functions. Then, using any of the above public key cipher systems, she encrypts $H(m)$ using her private key and attaches the result at the end of the message. Then she sends it off to Brad – probably she encrypts it first using Brad's public key.

Brad recovers the message m and, using Angelina's public key, he can decrypt s. Of course, he had preferred another contract with her, one that she would probably not sign. So he wishes to change the contract to another version m'. The problem is he has to do it in such a way that $s = H(m')$, and that is exactly what is difficult with a hash function.

Signatures using hash functions are a special case of error-detecting codes. But they are used here in a completely different way. It would be very stupid to use a usual error-detecting code as a hash function, because it is usually easy to produce collisions, i.e., given m, find an m' such that $H(m) = H(m')$.

Designed in the early 90's, the Message Digest MD5 is still very much used. Typically when downloading files, often the MD5-checksum is provided with it, so that one can check if the file was corrupted during the transfer. So here it is used rather like a error-detecting code, but it is also needed for checking in nobody has altered your file for instance by adding a virus into it.

Also Unix-like systems used it for storing passwords: Your password can not be written down as such on the computer otherwise it would be easy to gain access without permission. So instead only $H(m)$, where m is the password is stored. When entered a password m', the computer checks if $H(m')$ agrees with the stored $H(m)$. When they agree it is unlikely that m' is different from m.

Here is how it works in principles. MD5 starts by splitting up the input m into blocks of 128 bits. Each of these blocks is itself split up into four blocks of 32 bits, say a, b, c, and d. As in the Feistel ciphers, one proceeds now though 16 rounds, each defined by a rule like

$$a' = d, \quad d' = c, \quad c' = b, \quad \text{and} \quad b' = F(a, b, c, d)$$

where F is some non-linear functions (which moreover changes from round to round). After these 16 rounds the result is concatenated to a 128 bit integer. All these integers from the different 128 bit block are in the end added modulo 2^{128}, the result is the message digest $H(m)$.

For example, here is a pair of an input[18] and a corresponding output

I know that the human being and the fish can coexist.
`ce5ee19205f9ea24f94f1037c83e3ca7`

The value of the hash function if written in hexadecimal notation, i.e., 'a' represents the hexadecimal digit 10, 'b' is 11, ..., 'f' stands for 15. If we change just a little bit the message m, we get a completely different digest:

I knew that the human being and the fish can coexist.
`a01c29091ee0c3d06d5c86d7e0895ade`

First problems on this hash functions MD5 were reported in 1995 and, in 2007, Arjen Lenstra and his collaborators have found a way of producing collisions. It is now not considered secure any more to use MD5 for signatures; one should use the successors SHA-1 and SHA-2.

They are similar algorithms but with output of up to 512 bits and the functions F involved are quite a lot more complicated. But again, it might well be that serious flaws will be discovered soon in them. The new SHA-3 was defined in 2012.

For example, the two messages above using the 512 bit version of SHA-2 give the digest

```
36411fb9b8db68cc14b7e9f94c01f0278dc58d69208b6ae07dc08791897c4785
a5c518a8e3a6e55facf439e8a5f360530045e321aa321b8d3b7288c18e3eee3f
```

and

```
0050ebf89aa35dea7841aa3d9853f250dda9054a8b432d208b4da7255f640019
78c91f9181809bfd4e7f375c86155682a34511ce83da7d78342a658e7f524859
```

[18] A bushism.

Problem Sheets

Questions for chapter I

Throughout these problems we will often use the following transcription of letters as numbers.

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
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

The indication in [] refers to the section in the text to which the question is related to.

1). [I.1] Consider the following code

a $= 0 \mapsto 0000000$	e $= 4 \mapsto 1111000$
b $= 1 \mapsto 1000000$	f $= 5 \mapsto 1111100$
c $= 2 \mapsto 1100000$	g $= 6 \mapsto 1111110$
d $= 3 \mapsto 1110000$	h $= 7 \mapsto 1111111$

a) Encode the message "cafe".

b) Decode the received message 1000000111100000000001100000011111111.

c) Try to decode the message 1110000000010011000001011111000000000.

d) Give an example showing that a single error in the transmission over the noisy channel may give another message.

2). [I.2] The **binary repetition code** of length n is the code in which the only two codewords are $\mathbf{0} = (0, 0, \ldots, 0)$ and $(1, 1, \ldots, 1)$. What are the parameters $[n, k, d]$ for this code? What is the largest number of errors per codeword that it will correct?

3). [I.2] Let $n \geqslant 2$. The **binary even-weight code** of length n consists of all words in V_n that have even weight. Find the parameters $[n, k, d]$ for this code.

4). [I.2] Use the Hamming code (I.2)

a = 0000 ↦ 0000000	g = 0110 ↦ 1100110	l = 1011 ↦ 1010101
b = 0001 ↦ 1110000	h = 0111 ↦ 0010110	m = 1100 ↦ 1000011
c = 0010 ↦ 1001100	i = 1000 ↦ 1101001	n = 1101 ↦ 0110011
d = 0011 ↦ 0111100	j = 1001 ↦ 0011001	o = 1110 ↦ 0001111
e = 0100 ↦ 0101010	k = 1010 ↦ 0100101	p = 1111 ↦ 1111111
f = 0101 ↦ 1011010		

to encode the message "hello". Then decode

1100110 0001011 0101111 0111100 1001101 1001111 0111100 1101001

using the nearest-neighbour decoding scheme.

5). [I.2] Let C be a code of minimum distance d. Let $\mathbf{w}$ be a received word. Suppose $\mathbf{c}$ is a codeword such that

$$d(\mathbf{w}, \mathbf{c}) \leqslant \frac{d-1}{2}.$$

Prove that $\mathbf{c}$ is the unique closest codeword to $\mathbf{w}$.

6). [I.2] **Extending a code**. Given a $[n, k, d]$-code C, add a 0 or 1 to the end of every codeword to form a code $\hat{C}$ of length $n+1$ in which every codeword has even weight. (So we add a 0 to every codeword that already has even weight, and a 1 to every codeword of odd weight.) Show that the minimum distance of $\hat{C}$ must be even. What can you say about the parameters of $\hat{C}$ (a) if d is even, (b) if d is odd?

What is $\hat{C}$ if $C = V_n$?

7). [I.2] **Shortening a code**. First method. Given a $[n, k, d]$-code C with $d \geqslant 2$, delete the last symbol from every codeword to get a code C^* of length $n-1$. What can you say about the parameters of C^*? Why must we have $d \geqslant 2$? When might C^* be more useful than C?

8). [I.2] **Shortening a code**. Second method. Given a $[n, k, d]$-code C, we can form two new codes C_0 and C_1. The code C_0 is the set of all codewords in C that end with a zero, which is then removed. Similarly C_1 is the set of all codewords that ended in 1 but with this 1 removed. Both codes C_0 and C_1 are of length $n-1$. What can you say about the parameters of C_0 and C_1?

9). [I.3]

a) Show that there exists a $[5, 2, 3]$-code, but no $[5, 3, 3]$-code.

b) Recall the Hamming's code is a $[7, 4, 3]$-code. Show that there are no $[6, 4, 3]$-codes, no $[7, 5, 3]$-codes, and no $[7, 4, 4]$-codes.

c) Show that a 2-error-correcting code in V_{10} must have $k \leqslant 4$.

10). [I.5] Determine which of the following are linear codes.

a) The trivial code $C = \{0\}$.

b) The full code $C = V_n$.

c) The code in exercise 1).

d) The binary repetition code in exercise 2).

e) The even weight code in exercise 3).

For each that is linear determine the dual code $C^{\perp}$.

11). [I.5] Let C be a linear code. Let G be a generator matrix and let H be a parity check matrix for C.

a) Is the extended code $\hat{C}$ defined in exercise 6) also linear?

b) Is the shortened code C^* defined in exercise 7) also linear?

c) Which of the shortened codes C_0 and C_1 defined in exercise 8) is linear ?

If the new code is linear in any of the above questions, describe how one can obtain generator and parity check matrices for the new code from G and H.

12). [I.5] Prove that, in a linear code, either all the codewords have even weight or exactly half have even weight and the other half have odd weight. What, if anything, can you say about the parity of the minimum distance of the code in each case?

13). [I.5] Let C_1 and C_2 be two linear codes with parameters $[n, k_1, d_1]$ and $[n, k_2, d_2]$. Let C be the code consisting of all vectors of the form $(\mathbf{x}, \mathbf{x} + \mathbf{y})$ in V_{2n} such that $\mathbf{x} \in C_1$ and $\mathbf{y} \in C_2$. Prove that C is linear with parameters $[2n, k_1 + k_2, d]$ where $d = \min(2\,d_1, d_2)$.

14). [I.5] Suppose there exist linear codes with parameters $[n_1, k, d_1]$ and $[n_2, k, d_2]$ and k-row generator matrices G_1 and G_2. Let G be the matrix $\begin{pmatrix} G_1 & G_2 \end{pmatrix}$ obtained by juxtaposing G_1 and G_2. What can you say about the parameters of the code with generator matrix G?

15). [I.6] For each of the following generator matrices, find a generator matrix in standard form for the same code if this is possible, or for an equivalent code otherwise. Write down the corresponding parity-check matrix, and find the dimension and minimum distance of each code. Are the codes generated by G_4 and G_5 (a) the same, (b) equivalent?

$$G_1 = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \end{pmatrix} \quad G_2 = \begin{pmatrix} 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 \end{pmatrix} \quad G_3 = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 \end{pmatrix}$$

$$G_4 = \begin{pmatrix} 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 \end{pmatrix} \quad \text{and} \quad G_5 = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{pmatrix}$$

16). [I.7] For each of the following parity-check matrices, construct the syndrome look-up table (the table of error syndromes against coset leaders) for the code that they define. Show how you would correct and decode the received words 0000111, 0001110, 1111111, 1101011, 0110111 and 0111000.

$$H_1 = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix} \qquad H_2 = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}$$

17). [I.8] Find the dimension and minimum distance of each code in the previous problem, and calculate P_{err} for it.

18). [I.8] Compute P_{err} for the binary repetition code of length $n = 3$ and compare it to the Hamming code Ham(3). Do the same for a linear $[5, 2, 3]$-code.

19). [I.9] Show that there exists a 1-error correcting linear code of length n if $k < n - \log_2(n)$,

20). [I.9] (hard) **The Plotkin bound**. Let C be a linear code with parameters $[n, k, d]$ such that $n < 2\,d$. Then we have the following inequality

$$2^k \leqslant 2 \cdot \left\lfloor \frac{d}{2d - n} \right\rfloor .$$

Prove this by comparing an upper bound and a lower bound for the sum

$$\sum_{\mathbf{x},\mathbf{y} \in C} d(\mathbf{x}, \mathbf{y}) .$$

21). [I.10] Write down the parity-check matrix for the Hamming code Ham(4) of length 15. Construct the syndrome look-up table and find the codeword closest to the following words: 010100101001000, 111000111000111, and 110011100111000.

22). [I.11] Prove that the Reed-Muller code $R(1, 3)$ is a self-dual $[8, 4, 4]$-code. A code is called **self-dual** if $C^{\perp} = C$.

23). [I.12] Which of the following codes are cyclic?

a) All of V_n.

b) The very trivial code $C = \{\mathbf{0}\}$.

c) The code $C = \{\mathbf{0},\ (1, 0, 0),\ (0, 1, 0),\ (0, 0, 1)\}$.

d) The code $C = \{(1, 1, 0), (0, 1, 0), (0, 1, 1), (1, 0, 1), (0, 0, 1)\}$.

e) The code of all words in V_n whose first coordinate is different from it second coordinate. $(n > 2.)$

f) The code of all words of even weight in V_n.

g) The code of all words of odd weight in V_n.

h) The binary repetition code $C = \{(0, 0, \ldots, 0), (1, 1, 1 \ldots, 1)\}$ of length n.

i) A linear $[5, 2, 3]$-code.

24). [I.13] Polynomials over $\mathbb{F}_2$.

a) Show that the polynomial $1 + X^2$ in $\mathbb{F}_2[X]$ factors, while the polynomials $1 + X + X^2$, $1 + X + X^3$ and $1 + X^2 + X^3$ cannot be factored.

b) Compute the remainder when dividing $\mathbf{f}(X)$ by $\mathbf{g}(X) = 1+X+X^2$ for $\mathbf{f}(X)$ one of the following polynomials: $1+X+X^4$, $1+X^3+X^4$, and $1+X+X^2+X^3+X^4$. And conclude that all three $\mathbf{f}(X)$ are irreducible.

c) Compute the quotient and remainder when dividing $X^{21} + 1$ by $1 + X + X^2 + X^4 + X^6$. Show that $1 + X + X^4$ divides $X^{15} - 1$ and find the full factorisation of the latter.

d) Explain how one can use repeatedly lemma I.16 to find the greatest common divisor of two polynomials $\mathbf{g}_1(X)$ and $\mathbf{g}_2(X)$. (Euclid's algorithm, see II.6.2). Use this to find the greatest common divisor of $1 + X + X^2 + X^4 + X^6$ and $1 + X^2 + X^4 + X^5 + X^6$.

25). [I.13] For each cyclic code in the list of exercise 23) give the generator polynomial.

26). [I.13] Let C_1 and C_2 be two cyclic codes of length n. Show that $C_1 \cap C_2$ and $C_1 + C_2 = \{\mathbf{x} + \mathbf{y} \,|\, \mathbf{x} \in C_1, \mathbf{y} \in C_2\}$ are cyclic codes, too. Find their generator polynomials in terms of the generator polynomial $\mathbf{g}_1$ and $\mathbf{g}_2$ of C_1 and C_2 respectively.

27). [I.13] How many linear cyclic codes of length 7 exist? How many of length 15? Determine all values of k such that there exists a linear cyclic code of length 21 and dimension k.

28). [I.14] Let $\mathbf{g}(X) = (1 + X)(1 + X + X^3)$ be the generator polynomial of a cyclic code C of length 7. Write down a generator matrix and a parity-check matrix for C. Determine its parameters $[n, k, d]$.

29). [I.15] Let C be the cyclic code of length 63 generated by $\mathbf{g}(X) = 1 + X + X^6$.

a) Find the parameters of C. [Hint: It is easy to see that $d \leqslant 3$ and it is fairly difficult to show that $d > 2$. You may use without proof that $X^{63} - 1$ is the smallest polynomial of the forms $X^k - 1$ that is divisible by $\mathbf{g}(X)$.

b) Find the closest codeword to

$$\mathbf{x} = (0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, \ldots, 0)$$

(all the coordinates in the dots are 0).

30). [I.15] Let C be the 'CRC-16-CCITT' code generated by the polynomial $X^{16}+X^{12}+X^5+1$. Encode the message 101101 as described in the final example of section I.15.

Problems for chapter II

31). [II.2] Decipher the following ciphertext[19], given that it was obtained by applying an affine cipher to a passage of English text. List the encryption mapping (or, if you prefer, the decryption mapping) in full, and explain how you broke it.

XD OBNYREM REM V CBMHT VU YTNU UL URJT NTSRGRUT AQVHOUN.

32). [II.2] Decipher the following ciphertext, given that it was obtained by applying a monoalphabetic cipher (not an affine cipher!) to a passage of English text. List the encryption mapping in full and explain how you broke it.

CWT VIFTDBOTBC NDINIAMRA GTDT DTYTQCTS PK CWT RMPIED ANIZTAOMB MA XDDTRTFMBC CI CWT NDIPRTOA CWMC CWT RXPDMDK ATDFXQT XA UMQXBV.

33). [II.2] Decipher the following English translation of a quote by François de la Rochfoucauld, encrypted with a affine cipher.

GXJ FPC JV VESVFY PCXYGVI YX RVVS XTI DVFIVY BQ JV GPOV KVVC TCPKMV YX RVVS BY XTIDVMOVD?'

34). [II.3] The following quote by Einstein was encrypted with a Vigenère cipher.

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

Explain why you would guess that the keylength is 2. Then assuming this, break the code.

35). [II.3] (tedious) Find the key used to encrypt the 'Dead Parrot' in section II.3.

36). [II.6.2] Find an integer b such that $b \cdot a \equiv 1 \pmod{101}$ when $a = 2$, 3 or 23.

37). [II.6.3] Using the fast exponentiation method compute 3^{526} modulo 527 on your calculator. Prove that 527 is not a prime number. If you have access to access to a computer program[20] that can handle large integer, use the same idea to prove that $2^{32} + 1$ is not prime.

[19] From a national newspaper, 24th February 1988

[20] `maple` and `mathematica` can do this of course, the free software `pari-gp` available at `http://pari.math.u-bordeaux.fr/` is much better for number theory and the free software `sage` available at `http://www.sagemath.org` is very good for cryptography. All of them have fast exponentiation already in them. Suppose you want to compute 3^{17} modulo 31. In `pari`, you can type '`Mod(3,31)^17`', in `sage` it is '`power_mod(3,17,31)`', in `maple` you use '`3 &^ 17 mod 31`' and in `mathematica` one should write '`PowerMod[3, 17, 31]`'.

38). [II.6.4] Find a primitive element modulo 23.

39). [II.7] In a miniature version of RSA, a user has public key $(N, e) = (3599, 31)$. Use the Euclidean Algorithm to find the private key. (You shouldn't need a calculator for this question – assuming you can factorise $x^2 - 1$!)

40). [II.7] In the sequel we will use the following encoding of messages. As before the characters 'a' up to 'z' correspond to the integers 0 up to 25, but now we add the following symbols as in example in section II.8.

space	apostrophe	period	comma	question mark
26	27	28	29	30

In a miniature version of RSA someone has encrypted a message with the public key $(N, e) = (85, 43)$. Find the private key and decode the message

2, 59, 66, 59, 0, 20, 64, 52, 66, 59, 28, 44, 66, 59, 44, 66, 59, 0, 72, 56, 44, 12.

41). [II.7] (in groups) Each participant should set up a secret and a public key as in a mini-version of the RSA ciphersystem. To make the computations feasible one should take the prime factors p and q between 30 and 100. Then the public key is given to one other member of the group who will then encrypt a short message (not more than 20 letters) with it.

To encrypt, we use now blocks of two letters (or symbols). If a and b are the numeric value of the two symbols in the block, we translate the block to the plaintext $m = a + b \cdot 31$. If the length of the full text is not even, just add a space to it in the end. For instance, the text "good." is transcribed to 440, 107, 834.

The holder of the secret key can then decrypt and read the message. It is also a good exercise to add a last round where everyone tries to break one ciphertext, only knowing the public key.

42). [II.8] This is a baby version of the Elgamal encryption. We choose the prime $p = 101$. The integer 2 is a primitive root modulo p. As a private key, you choose $d = 96$. Compute the public key and decrypt the message

$(68, 0), (33, 28), (2, 90), (79, 52), (32, 66), (75, 80), (69, 63), (87, 0), (87, 51), (53, 91), (9, 0), (93, 69), (100, 5), (74, 91), (2, 57), (58, 2), (42, 38), (33, 96), (43, 86)$

where we used the same conventions as in previous exercises. Explain why even this baby version is better than a monoalphabetic cipher, assuming the attacker knows neither the public nor the private key.

43). [II.9] Amy and Bryan would like to agree on a secret key $1 < n < 1000$. Both choose private keys: $d_A = 513$ and $d_B = 33$ modulo $p = 1009$, respectively. The smallest primitive element modulo p is $a = 11$. Use the Diffie-Hellmann to compute the secret key for both Amy and Bryan and show that they agree.

44). [II.10] Azaliah needs to send the two-digit number $m = 12$ to her mate Belteshazzar[21]. They agree to use the prime $p = 103$. She chooses the secrect key $d_A = 67$ and Belteshazzar decided that he will use his secret key $d_B = 5$. Use the no-key protocol to send the message m.

45). [II.11.2] You wish to add a signature to your message using the Digital Signature Algorithm. Suppose the hash-function of your message text is $s = 721$. Your private Elgamal key is $(p, a, d) = (1019, 2, 17)$. Compute the signature (s, r, t) and verify that the receiver, knowing your public key can verify the signature.

46). [II.11.2] Alanis sends to Bruce the signature $(s, r, t) = (423, 32, 231)$. You know that her public key is $(p, a, e) = (1019, 2, 479)$. Solve the discrete logarithm problem for $r = a^k$, and find the private key of Alanis.

[21] biblical name meaning 'who lays up treasures in secret'

Bibliography

[1] Henry Beker and Fred Piper, *Cipher systems: the protection of communications*, Northwood Bookds, London, 1982.

[2] Cryptograms.org, *Frequency of letters*, `http://www.cryptograms.org/letter-frequencies.php`, visited January 2013.

[3] Markus Grassl, *Code Tables: Bounds on the parameters of various types of codes*, `http://www.codetables.de/`, the table `http://iaks-www.ira.uka.de/home/grassl/codetables/BKLC/Tables.php?q=2&n0=1&n1=20&k0=1&k1=20` is the most relevant for this module; visited January 2013.

[4] Darrel Hankerson, Greg A. Harris, and Peter D. Johnson, Jr., *Introduction to information theory and data compression*, second ed., Discrete Mathematics and its Applications (Boca Raton), Chapman & Hall/CRC, Boca Raton, FL, 2003.

[5] Raymond Hill, *A first course in coding theory*, Oxford Applied Mathematics and Computing Science Series, The Clarendon Press Oxford University Press, New York, 1986.

[6] Gareth A. Jones and J. Mary Jones, *Elementary number theory*, Springer Undergraduate Mathematics Series, Springer-Verlag London Ltd., London, 1998.

[7] David Joyner and Robert Miller, *sage and Coding Theory*, 2008, `http://sage.math.washington.edu/home/wdj/cookbook/coding-theory/sage-coding-cookbook.pdf`.

[8] Neal Koblitz, *A course in number theory and cryptography*, Graduate Texts in Mathematics, vol. 114, Springer-Verlag, New York, 1987.

[9] T. W. Körner, *Coding and Cryptography*, `http://www.dpmms.cam.ac.uk/~twk/` visited January 2013, 1999.

[10] RSA Laboratories, *The rsa factoring challenge*, `http://www.rsa.com/rsalabs/node.asp?id=2092` visited January 2013.

[11] San Ling and Chaoping Xing, *Coding theory*, Cambridge University Press, Cambridge, 2004, A first course.

[12] Nitin Saxena Manindra Agrawal, Neeraj Kayal, *PRIMES is in P*, Annals of Mathematics **160** (2004), no. 2, 781–793, available at `http://www.jstor.org/stable/3597229`.

[13] Minh Van Nguyen, *Number Theory and the RSA Public Key Cryptosystem*, 2008, https://bitbucket.org/mvngu/numtheory-crypto/downloads/numtheory-crypto.pdf.

[14] The PARI Group, Bordeaux, *PARI/GP, version* 2.3.5, 2010, available from http://pari.math.u-bordeaux.fr/.

[15] Simon Singh, *The Code Book: The Science of Secrecy from Ancient Egypt to Quantum Cryptography*, Anchor, 2000.

[16] William Stein, *Elementary number theory: primes, congruences, and secrets: A computational approach*, Undergraduate Texts in Mathematics, Springer, New York, 2009, available online for free at http://modular.math.washington.edu/ent.

[17] William Stein et al., *Sage Mathematics Software (Version 4.6)*, The Sage Development Team, 2010, http://www.sagemath.org.

[18] Peter Symonds, *Coding theory*, notes for MATH32031 available at http://www.maths.manchester.ac.uk/~pas/, 2009.

[19] International Telecommunication Union, *Transmission Systems and Media, Digital Systems and Networks (g. 704)*, 1998, pp. 1–45.

[20] Björn von Sydow, *Mono-, Bi and Trigram Frequency for English*, no longer available online.

[21] Dominic Welsh, *Codes and cryptography*, Oxford Science Publications, The Clarendon Press Oxford University Press, New York, 1988.

[22] Phil White, *Error Correction Codes: FAQs*, http://web.archive.org/web/19990117020045/http://members.aol.com/mnecctek/faqs.html visited January 2013.

[23] Wikipedia, *Advanced Encryption Standard*, http://en.wikipedia.org/wiki/Advanced_Encryption_Standard visited January 2013.

[24] Wikipedia, *Cyclic Redundancy Check*, http://en.wikipedia.org/wiki/Cyclic_redundancy_check visited January 2013.

[25] Wikipedia, *Data Encryption Standard*, http://en.wikipedia.org/wiki/Data_Encryption_Standard visited January 2013.

[26] Wikipedia, *Hamming's code*, http://en.wikipedia.org/wiki/Hamming%287%2C4%29 visited January 2013.