MATH320
Mathematical Cryptography
2025/26

Dr. Dan Fretwell
(d.fretwell@lancaster.ac.uk)

Lancaster University

Contents

1 Classical Cryptography: A brief history

Information is a valuable asset. However, it is often necessary to keep certain pieces of information secret from others. This has been the case ever since the dawn of time!

Mathematical Cryptography concerns the use of mathematical tools and structures in order to provide data security. This is not just an interesting and exciting topic, but is a crucial area of research in our current digital age, since more and more sensitive data is being captured than ever before.

In this chapter we first begin by looking to the past and studying classical encryption methods.

1.1 The Caesar cipher: What would Julius Caesar do?

Cryptography is the "art of secret writing". In modern language, the main idea is that one can **encrypt** messages taken from a set M by applying an **encryption map**, $e : M \to C$ (with C some other set). There are of course many choices for e and C, but we wish to choose them so that it will be **infeasible** to reconstruct the **plaintext** message $m \in M$ purely from the **ciphertext** message $c = e(m) \in C$. We also want to choose e so that it is **efficient** to compute $e(m)$.

How would friends be able to read the message? The sender has been clever...the encryption map $e = e_k$ secretly depends on the knowledge of a **key** $k \in K$, as does the corresponding **decryption map** $d_k : \mathrm{Im}(e_k) \to M$ satisfying $d_k \circ e_k = \mathrm{id}_M$. Anyone knowing the key can compute both maps e_k and d_k efficiently.

We usually assume that the sets M, C and K are finite, but they need not be. For most of the first half of this course we will take $M = C = \{\mathtt{A},\mathtt{B},\mathtt{C}...,\mathtt{Z}\}$ to be the usual alphabet, but again they need not be. We will also often identify letters of the alphabet with elements of $\{0, 1, ..., 25\}$ via:

$$\mathtt{A} \longrightarrow 0 \qquad \mathtt{B} \longrightarrow 1 \qquad ... \qquad \mathtt{Y} \longrightarrow 24 \qquad \mathtt{Z} \longrightarrow 25.$$

If the sender wishes a friend to read their messages then they can be told the key k. This will then allow them to **decrypt** messages by computing $d_k(c) = (d_k \circ e_k)(m) = \mathrm{id}_M(m) = m$.

A general principle of Kerckhoffs is that the security of the above process should depend **only** on keeping the key secure (this seems obvious, right?). Later we'll see that modern Cryptography does not follow this principle!

A silly example:

It will soon be Eve's birthday, and she has been strongly hinting all year that she would like a new MacBook Pro...but it has to be the newest 2024 model with 16-inch screen, 1TB SSD and all the other bells and whistles.

Alice is a seemingly devoted friend and has bought her one, but wants to make sure Eve's (only) other friend Bob hasn't done the same. She types out a text to Bob (Alice always writes in capitals since she is constantly in a state of extreme emotion):

OMG, I SPENT LIKE THOUSANDS ON THIS MACBOOK FOR EVE. I WAS LIKE WHAT THE HELL AND THE GUY IN THE STORE WAS LIKE YOU NEED A NEW FRIEND. DID YOU LIKE GET ONE TOO?

However, Alice wants to make absolutely sure that Eve will not see this message, otherwise the surprise will be spoiled! She decides to use a classical technique to encrypt the message.

Definition 1.1. *The **Caesar cipher** with key $k \in \{0, 1, ..., 25\}$ is the cipher with encryption/decryption maps as follows:*

If $m \in M$ is a plaintext letter then the ciphertext letter $c = e_k(m)$ is given by shifting m forwards k places in the alphabet (wrapping around to A *if reaching* Z*).*

We decrypt the ciphertext letter $c \in M$ (i.e. compute the corresponding plaintext letter $m = d_k(c)$) by shifting c backwards k places in the alphabet (wrapping around to Z *if reaching* A*).*

Example 1.2. *If Alice chooses her key to be $k = 3$ then she gets the following ciphertext:*

RPJ, L VSHQW OLNH WKRXVDQGV RQ WKLV PDFERRN IRU HYH. L ZDV OLNH ZKDW WKH KHOO DQG WKH JXB LQ WKH VWRUH ZDV OLNH BRX QHHG D QHZ IULHQG. GLG BRX OLNH JHW RQH WRR?

If we identify letters with numbers as earlier then we can write the encryption/decryption maps mathematically as follows:

$$\begin{aligned} e_k : \{0, 1, ..., 25\} &\longrightarrow \{0, 1, ..., 25\} \\ m &\longmapsto c \equiv m + k \bmod 26 \\ d_k : \{0, 1, ..., 25\} &\longrightarrow \{0, 1, ..., 25\} \\ c &\longmapsto m \equiv c - k \bmod 26 \end{aligned}$$

Exercise 1.3. *Check that the mathematical encryption/decryption maps satisfy $d_k \circ e_k = \mathrm{id}$.*

The Caesar cipher was used by emperor **Julius Caesar** in Ancient Rome to communicate battle plans between various camps. This is a very simple method of encryption by today's standards, since it is very easy to brute force all possible 26 shifts and hence decrypt any ciphertext. However, back then this was a completely new thing and so was perfectly secure (also few people could read or write).

1.2 Substitution ciphers: What would Mary Queen of Scots do?

The Caesar cipher shifts plaintext letters by a fixed amount in the alphabet. Mathematically speaking, a sequence of plaintext values $m_1, m_2, ..., m_t$ are encrypted to a sequence of ciphertext values $c_1, c_2, ..., c_t$ via the map $c_i \equiv m_i + k \bmod 26$, where $k \in \{0, 1, ..., 25\}$ is the key.

However, as mentioned above this method clearly provides very little security since it is extremely easy to check all 26 possible keys...even on paper.

There are many ways to generalise the Caesar cipher. One is to notice that the map:

$$\begin{aligned} \{0,1,...,25\} &\longrightarrow \{0,1,...,25\} \\ x &\longmapsto x+k \bmod 26 \end{aligned}$$

is a simple example of bijection. In general, we could have used any permutation of this set.

Recall that for each set X we have a group S_X of permutations of X, i.e. bijections $\sigma : X \to X$. The group operation is composition. In particular, if $X = \{x_1, x_2, ..., x_n\}$ is finite then we write S_n for the group S_X and elements $\sigma \in S_X$ act by $\sigma(x_i) = x_{\sigma(i)}$. So we can think of S_{26} as the group of permutations of each of the sets $M = C = \{$A,B,C...,Z$\}$ and $\{0, 1, ..., 25\}$.

Definition 1.4. *The* ***monoalphabetic substitution cipher*** *with key $\sigma \in S_{26}$ is the cipher with encryption/decryption maps:*

$$\begin{aligned} e_\sigma : \{0,1,...,25\} &\longrightarrow \{0,1,...,25\} \\ m_i &\longmapsto c_i = \sigma(m_i) \\ d_\sigma : \{0,1,...,25\} &\longrightarrow \{0,1,...,25\} \\ c_i &\longmapsto m_i = \sigma^{-1}(c_i). \end{aligned}$$

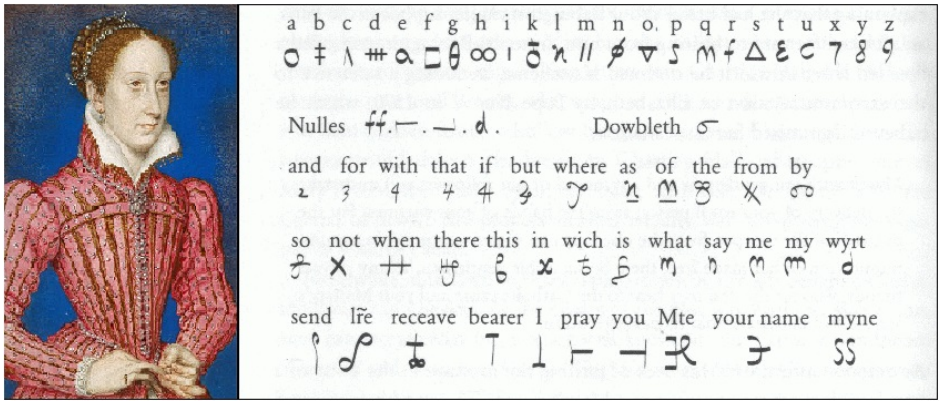
Example 1.5. *If Alice chooses her key to be $\sigma \in S_{26}$ with cycle decomposition:*

$$\sigma = (0\ \ 1)(2\ \ 3)(4\ \ 5)...(24\ \ 25)$$

(i.e. we swap $0 \longleftrightarrow 1$, $2 \longleftrightarrow 3$ etc) then she gets the following ciphertext:

PNH, J TOFMS KJLF SGPVTBMCT PM SGJT NBDAPPL EPQ FUF. J XBT KJLF XGBS SGF GFKK BMC SGF HVZ JM SGF TSPQF XBT KJLF ZPV MFFC B MFX EQJFMC. CJC ZPV KJLF HFS PMF SPP?

Such ciphers have been incredibly popular throughout history. One famous use of substitution ciphers was by **Mary Queen of Scots** in Tudor times to communicate a plot to assassinate Queen Elizabeth I with her conspirators (although this is an example with $M \neq C$).



Example 1.6. *The Caesar cipher was insecure because we could brute force all 26 possible keys quickly. How many keys does the monoalphabetic substitution cipher have?*

We have $|S_{26}| = 26!$, since there are 26 possibilities for $\sigma(0)$, and for each there are 25 possibilities for $\sigma(1)$, then for each there are 24 possibilities for $\sigma(2)$ etc.

Note that $26! = 403291461126605635584000000$, an astronomical number!

Given the huge number of keys you might believe that the monoalphabetic substitution cipher is secure. However, given enough ciphertext it's possible to use simple statistical methods to decrypt.

The idea is simple...since we always send a given plaintext letter to the same ciphertext letter, we are not changing the **frequencies** of each letter. Couple this with the fact that certain letters in English are used more than others and we have an attack!

The most common letters used in the English language are ETAOINSHRDLU (in this order). For example, if you take any sufficiently large amount of English text, you'll find that roughly 12.7% of the letters are the letter E. This means that even though we don't know what $\sigma(\texttt{E})$ is, we can take a guess since it's likely to be the letter that appears roughly 12.7% of the time in the ciphertext message.

See the Appendix for a full table of expected letter frequencies in the English language.

Example 1.7. *Suppose we intercept the following ciphertext message:*

SGF OQPAKFN XJSG SGF NPMPBKOGBAFSJD TVATSJSVSJPM DJOGFQ JT SGBS KJLF KFSSFQT BKXBZT HFS TFMS SP KJLF KFSSFQT. SGF EQFRVFMDJFT PE KFSSFQT JM SGF OKBJMSFWS BQF OQFTFQUFC JM SGF DJOGFQSFWS.

We make a table of letter frequencies:

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
3	7	1	4	2	28	10	1	0	13	9	2	7	2	6	6	10	1	25	10	1	3	2	2	0	1

The most common letters in the ciphertext are F *and* S*. We guess that* $\texttt{F} = \sigma(\texttt{E})$ *and* $\texttt{S} = \sigma(\texttt{T})$ *and see what happens:*

TGE OQPAKEN XJTG TGE NPMPBKOGBAETJD TVATTJTVTJPM DJOGEQ JT TGBT KJLE KETTEQT BKXBZT HET TEMT TP KJLE KETTEQT. TGE EQERVEMDJET PE KETTEQT JM TGE OKBJMTEWT BQE OQETEQUEC JM TGE DJOGEQTEWT.

The letter G *is quite common and the word* TGE *could be* THE*. We make the guess that* $\texttt{G} = \sigma(\texttt{H})$*:*

THE OQPAKEN XJTH THE NPMPBKOHBAETJD TVATTJTVTJPM DJOHEQ JT THBT KJLE KETTEQT BKXBZT HET TEMT TP KJLE KETTEQT. THE EQERVEMDJET PE KETTEQT JM THE OKBJMTEWT BQE OQETEQUEC JM THE DJOHEQTEWT.

Continuing on in this fashion eventually gives the plaintext:

THE PROBLEM WITH THE MONOALPHABETIC SUBSTITUTION CIPHER IS THAT LIKE LETTERS ALWAYS GET SENT TO LIKE LETTERS. THE FREQUENCIES OF LETTERS IN THE PLAINTEXT ARE PRESERVED IN THE CIPHERTEXT.

Exercise 1.8. *Why should Alice stop using the word* LIKE *so often if she wants to use a monoalphabetic substitution cipher to communicate with Bob?*

1.3 The Vigenère cipher: What would secret Victorian lovers do?

The monoalphabetic substitution cipher is one way to generalise the Caesar cipher, but it suffers from insecurity due to the fact that it preserves letter frequencies. The Vigenère cipher is an example of a **polyalphabetic substitution cipher**, one where the key used changes from plaintext letter to plaintext letter.

Recall that the Caesar cipher encrypts via the map $e_k(m_i) \equiv m_i + k \bmod 26$. The idea of the Vigenère cipher is to make k **variable**.

Definition 1.9. *The **Vigenère cipher** with key $\boldsymbol{k} = (k_1, k_2, ..., k_n) \in \{0, 1, ..., 25\}^n$ is the cipher with encryption/decryption maps:*

$$\begin{aligned} e_{\boldsymbol{k}} : \{0, 1, ..., 25\} &\longrightarrow \{0, 1, ..., 25\} \\ m_i &\longmapsto c_i \equiv m_i + k_{i \bmod n} \bmod 26 \\ d_{\boldsymbol{k}} : \{0, 1, ..., 25\} &\longrightarrow \{0, 1, ..., 25\} \\ d_{\boldsymbol{k}} &\longmapsto m_i \equiv c_i - k_{i \bmod n} \bmod 26 \end{aligned}$$

In other words, we have multiple Caesar shifts and we apply them to plaintext letters in order (cycling around to k_1 when we reach k_n). We call the integer $n \geq 1$ the **key length**.

Exercise 1.10. *Show that the Caesar cipher is just the Vigenère cipher with key length $n = 1$.*

Example 1.11. *If Alice chooses $\boldsymbol{k} = (1, 2) \in \{0, 1, ..., 25\}^2$ as her key then she gets the following ciphertext:*

```
POH, K TRFPU NJMF VIQVUBPEU PP UJJU NCDDPQL HPT FXF. K XCT NJMF YICU VIG IGMN BPE VIG
HWZ KO VIG TVPTF YBU MKLG ZQV PFGE C OGX HSKFPE. FJF ZQV NJMF IFV PPF VPQ?
```

Note that by varying the shifts we can now have **different** plaintext letters being encrypted to the **same** ciphertext letter and also the **same** plaintext letters being encrypted to **different** ciphertext letters. For example, in the above example we see that the word `MACBOOK` is encrypted to `NCDDPQL`, demonstrating both points.

This destroys the letter frequencies, hence frequency analysis is useless!

Example 1.12. *To demonstrate this point, suppose that we instead used a longer key, say $\boldsymbol{k} = (1, 2, 3, 4, 5, 6) \in \{0, 1, ..., 25\}^6$ then the ciphertext and letter frequencies are:*

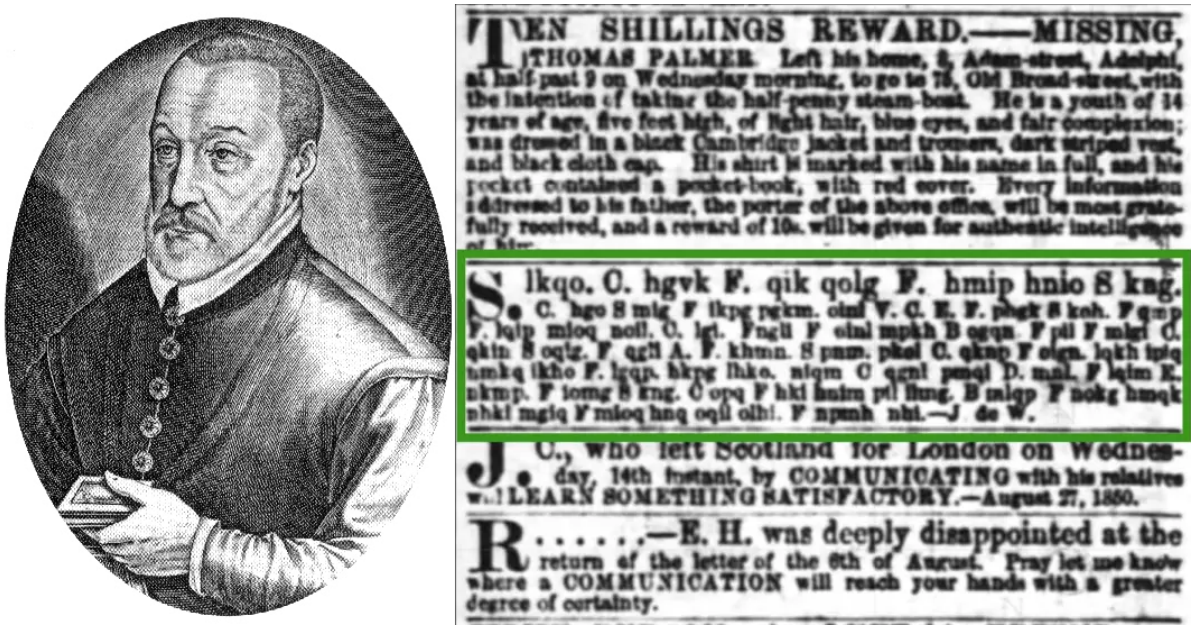
```
POJ, M XVFPW PNQF VKSZYBPGW TT UJLW RGDDRSP LPT HZJ. O XCV PNQF YKEY ZIG KIQR BPG XMK
HWB MS ZIG VXTXF YDW QOLG BSZ TFGG E SKX HUMJTE. FLH DUV NLOJ MFV RRJ ZPQ?
```

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

Note that we could instead choose a **keyword** as our key, and convert this to numbers in order to get our shifts. This makes it easier to communicate the key with your friend.

Example 1.13. *If Alice chooses keyword* `BOB` *then she would encrypt her message using the key* $\boldsymbol{k} = (1, 14, 1) \in \{0, 1, ..., 25\}^3$ *of length* $n = 3$.

The Vigenère cipher was a very popular cipher from the late 1500's through to the mid 1800's. In particular, it was often used in the Victorian era to allow secret lovers to communicate their affairs in public newspapers.



TEN SHILLINGS REWARD.——MISSING, THOMAS PALMER. Left his home, 5, Adam-street, Adelphi, at half-past 9 on Wednesday morning, to go to 75, Old Broad-street, with the intention of taking the half-penny steam-boat. He is a youth of 14 years of age, five feet high, of light hair, blue eyes, and fair complexion; was dressed in a black Cambridge jacket and trousers, dark striped vest, and black cloth cap. His shirt is marked with his name in full, and his pocket contained a pocket-book, with red cover. Every information addressed to his father, the porter of the above office, will be most gratefully received, and a reward of 10s. will be given for authentic intelligence of him.

S lkqo. C. hgvk F. qik qolg F. hmip hnio S kng. C. hgo S mlg F lkpg rgkm. olnl V. C. E. F. phgk S koh. F qmp F. kqip mloq noll. C. lgi. Fngll F olnl mpkh B ogqn F pil F mlgi C. qkin S oqlg. F qgll A. F. khmn. S pnm. phol C. qknp F olgn. lqkh lpiq nmkq ikho F. lgqn. hkpg lhko. mlqm C qgnl pmql D. mnl. F lqim E. nkmp. F iomg S kng. C opq F hkl lnnlm pil lhng. B mlgp F nokg hmqk nhkl mgiq F mloq hnq oqll olhl. F npmh nhl.—J do W.

J. C., who left Scotland for London on Wednesday, 14th instant, by COMMUNICATING with his relatives will LEARN SOMETHING SATISFACTORY.—August 27, 1850.

R . . . . . —E. H. was deeply disappointed at the return of the letter of the 6th of August. Pray let me know where a COMMUNICATION will reach your hands with a greater degree of certainty.

For a long time the Vigenère cipher was thought to be unbreakable and was often referred to as **le chiffrage indéchiffrable** (the indecipherable cipher). However, in the 1860's an attack on Vigenère was found by **Friedrich Kasiski** that worked quite well for long ciphertexts.

The Kasiski test tries to identify the key length n. The main idea is to observe that if repeated strings of letters in the plaintext are encrypted with the **same** parts of the key then the resulting ciphertext strings must also repeat.

Example 1.14. *If we encrypt* `THE THE THE THE` *with the key* $\boldsymbol{k} = (1, 2) \in \{0, 1, ..., 25\}^2$ *then we get* `UJF VIG UJF VIG`.

We see that the first and third instances of the word `THE` *were both encrypted to* `UJF`. *This was because they were both encrypted using the same sequence of shifts* $(1, 2)$.

The same is true for the second and fourth instances, but now the encrypted string is `VIG` *since these were instead encrypted with the sequence of shifts* $(2, 1)$.

Kasiski's idea was to look for repeated strings in the ciphertext. Assuming that these were encrypted with the same part of the key, the **spacings** would then reveal information about the key lengths!

Example 1.15. *We'll look at the following ciphertext, encrypted using an unknown Vigenère keyword:*

ZPGDL RJLAJ KPYLX ZPYYG LRJGD LRZHZ QYJZQ REPVM SWRZY RIGZH ZVREG KWIVS
SAOLT NLIUW OLDIE AQEWF IIYKH BJOWR HDOGC QHKWA JYAGG EMISR ZQOQH OAVLK
BJOFR YLVPS RTGIU AVMSW LZGMS EVWPC DMJSV JQBRN KLPCF IOWHV KXJBJ PMFKR
QTHTK OZRGQ IHBMQ SBIVD ARDYM QMPBU NIVXM TZWQV GEFJH UCBOR VWPCD XUWFT
QMOOW JIPDS FLUQM OEAVL JGQEA LRKTI WVEXT VKRRG XANI

If we look through the ciphertext we find lots of repeated trigrams:

Trigram	Ciphertext position	Distance	Factorisation
AVL	117, 258	141	$3 \cdot 47$
BJO	86, 121	35	$5 \cdot 7$
DLR	4, 25	21	$3 \cdot 7$
GDL	3.24	21	$3 \cdot 7$
LRJ	5, 21	16	2^4
MSW	40, 138	98	$2 \cdot 7^2$
PCD	149, 233	84	$2^2 \cdot 3 \cdot 7$
QMO	241, 254	13	13
VMS	39, 137	98	$2 \cdot 7^2$
VWP	147, 231	84	$2^2 \cdot 3 \cdot 7$
WPC	148, 232	84	$2^2 \cdot 3 \cdot 7$
ZHZ	28, 49	21	$3 \cdot 7$

Assuming that most of these were encrypted using the same part of the keyword, it seems likely that the key is of length $n = 7$ (note that this isn't an exact science...sometimes repeated trigrams will occur by chance, as is the case for AVL, LRJ *and* QMO*).*

Once you have a reasonable guess at the keyword length n, you can proceed in many ways (e.g. if n is small then you can simply brute force all possibilities). Over the next two lectures we'll see good statistical attacks that work in practice.

1.4 The One Time Pad: What would the president do?

At this point in the course, every cipher we have seen has some sort of weakness. A natural question might pose itself to you...does there exist a cipher that is **guaranteed** to be secure?

It turns out that we aren't too far from the answer to this question.

Definition 1.16. *If $\boldsymbol{m} = m_1, m_2, ..., m_t$ is a plaintext message then the **One Time Pad** with key $\boldsymbol{k} = (k_1, k_2, ..., k_t) \in \{0, 1, ..., 25\}^t$ is the cipher with encryption/decryption maps:*

$$\begin{aligned} e_{\boldsymbol{k}} : \{0, 1, ..., 25\} &\longrightarrow \{0, 1, ..., 25\} \\ m_i &\longmapsto c_i \equiv m_i + k_i \bmod 26 \\ d_{\boldsymbol{k}} : \{0, 1, ..., 25\} &\longrightarrow \{0, 1, ..., 25\} \\ d_{\boldsymbol{k}} &\longmapsto m_i \equiv c_i - k_i \bmod 26 \end{aligned}$$

In other words, we use the Vigenère cipher but with a key as long as the message.

Note that if the key is chosen uniformly at random then the One Time Pad is provably secure. The reason is as follows:

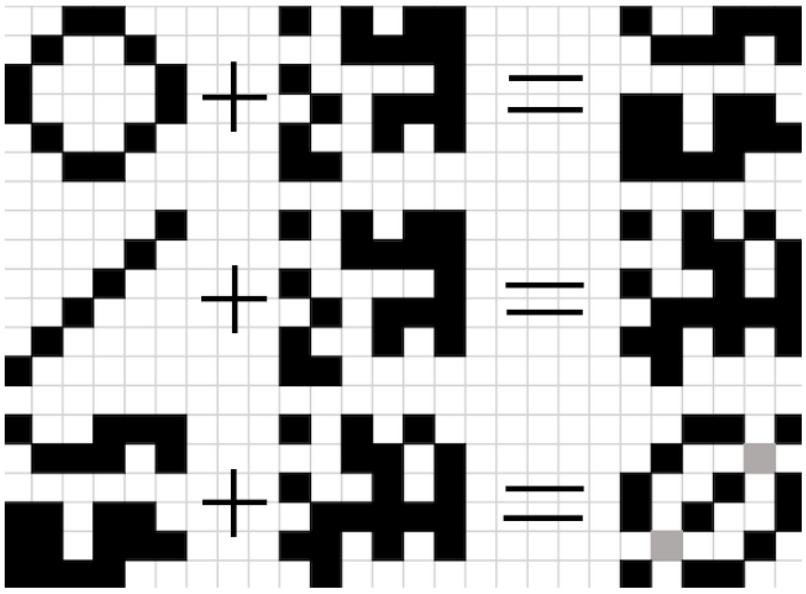
Example 1.17. *Suppose we receive the ciphertext* `AAA`*, encrypted using a One Time Pad chosen uniformly at random. Then the potential plaintexts* `CAT` *and* `DOG` *are equally likely, since the corresponding encryption keys had the same probability of being chosen by Alice.*

For the same reason, every string of length three is equally likely to be the plaintext. Because of this fact we cannot learn anything about the plaintext from the ciphertext.

Before we get too excited, the One Time Pad does come with some disadvantages:

- The key has to be the same length as the plaintext message, and so will be **huge**. Communicating this with your friends is quite **inefficient** and is hard to do **securely**!
- To have perfect security, the key needs to be generated **uniformly** at random. This is extremely hard to do, especially by a human!
- As the name implies, you should only ever use the key **once**. The reason for this is that if we use the **same** key $\mathbf{k}$ to encrypt plaintext messages $\mathbf{m}$ and $\mathbf{m}'$ then the corresponding ciphertext messages $\mathbf{c}$ and $\mathbf{c}'$ are given by $c_i \equiv m_i + k_i \bmod 26$ and $c'_i \equiv m'_i + k_i \bmod 26$. Subtracting gives $c_i - c'_i \equiv m_i - m'_i \bmod 26$, which is completely independent of the key!

  This is easier to demonstrate by picture:



  In the first two rows we add the same random bitstring to two pictures (working mod 2). In the third row we add the ciphertexts (same as subtracting mod 2) and get an amalgamation of the original pictures. The key has been completely removed, revealing the two pictures!

Historically, the One Time Pad was often used to encrypt phone calls between the presidents of the USA and the USSR. Random bit strings would be generated and sent securely via big diplomatic bags, printed on nitrocellulose sheets (so that they could be easily burned after use).

1.5 The Index of Coincidence

We have just seen that the Kasiski test can be used to determine the potential key length of a Vigenère ciphertext, by searching for repeated strings in the text. What do we do next?

Suppose that we have a Vigenère ciphertext message $\mathbf{c} = c_1c_2...c_t$, encrypted using key $\mathbf{k} = (k_1, k_2, ..., k_n)$. By definition of the encryption map we see that the letters $c_1, c_{1+n}, c_{1+2n}, ...$ were encrypted using the **same** shift k_1. Similarly, for each $1 \leq i \leq n$ we find that $c_i, c_{i+n}, c_{i+2n}, ...$ were encrypted using the **same** shift k_i.

Definition 1.18. *If $\boldsymbol{c} = c_1c_2...c_t$ is a Vigenère ciphertext message, encrypted using a key of length n then the strings:*

$$\begin{aligned}
\boldsymbol{s}_1 &= c_1 \quad c_{1+n} \quad c_{1+2n} \quad ... \\
\boldsymbol{s}_2 &= c_2 \quad c_{2+n} \quad c_{2+2n} \quad ... \\
&\vdots \\
\boldsymbol{s}_n &= c_n \quad c_{2n} \quad c_{3n} \quad ...
\end{aligned}$$

*are called the **partial ciphertexts** of $\boldsymbol{c}$.*

Example 1.19. *Returning to the Vigenère ciphertext in the previous section we find that the partial ciphertexts are:*

$$\begin{aligned}
\boldsymbol{s}_1 &= \texttt{ZLXRHRRHWLOEHDWEOKLILWVLHPHQBYNWHWFJULRXX} \\
\boldsymbol{s}_2 &= \texttt{PAZJZEZZITLWBOAMQBVUZPJPVMTIIMIQUPTIQJKTA} \\
\boldsymbol{s}_3 &= \texttt{GJPGQPYVVNDFJGJIHJPAGCQCKFKHVQVVCCQPMGTVN} \\
\boldsymbol{s}_4 &= \texttt{DKYDYVRRSLIIOCYSOOSVMDBFXKOBDMXGBDMDOQIKI} \\
\boldsymbol{s}_5 &= \texttt{LPYLJMIESIEIWQARAFRMSMRIJRZMAPMEOXOSEEWR} \\
\boldsymbol{s}_6 &= \texttt{RYGRZSGGAUAYRHGZVRTSEJNOBQRQRBTFRUOFAAVR} \\
\boldsymbol{s}_7 &= \texttt{JLLZQWZKOWQKHKGQLYGWVSKWJTGSDUZJVWWLVLEG}
\end{aligned}$$

Assuming the key is indeed of length $n = 7$ we find that $\boldsymbol{s}_1$ was encrypted using shift k_1, $\boldsymbol{s}_2$ by shift k_2, etc.

The question now is, how do we find the shifts? We could try and brute force all 26^7 possibilities, but this is not an easy task...even on a (standard) computer!

One way to proceed is to remember that a Caesar cipher is an example of a monoalphabetic substitution cipher...so that we **can** try to use frequency analysis on each of the strings $\mathbf{s}_i$. If we look for common letters in each $\mathbf{s}_i$ and make educated guesses then this might tell us the shifts used.

Example 1.20. *In* $\boldsymbol{s}_1$ *we see that the most common letters are* H, L, *and* W. *Guessing that* H *shifts back to* E *would imply that* L *shifts back to* I *and* W *shifts back to* T. *This seems like a pretty consistent guess, so perhaps* $k_1 = 4$.

However, this is not an exact science...as if we do the same for $\boldsymbol{s}_2$ *we find that the most common letters are* I, Z *and* P. *Guessing that any of these shift back to* E *leads to issues with the other two (one of them would shift back to either* V *or* X, *which seems unlikely).*

We'll see soon that a better method exists for deducing information about the shifts of the $\mathbf{s}_i$ if we know the key length. However, first we take a necessary detour.

Let's go back to the problem of finding the key length n. Given what we discovered above, a new method presents itself for doing this:

Pseudoalgorithm

1. Set potential key length to be $n = a$.
2. Find the partial ciphertexts $\mathbf{s}_1, \mathbf{s}_2, ..., \mathbf{s}_a$.
3. Run an interesting statistical test on each $\mathbf{s}_i$ to determine whether it **could** have been encrypted using a monoalphabetic substitution cipher, as opposed to being "random".
4. If successful then guess that the key length is a. Otherwise, increase by 1 and start again.

The question now is how to find **good** statistics that distinguish text generated by a monoalphabetic substitution cipher from "random" text. The Index of Coincidence is one such statistic.

Definition 1.21. *The* ***Index of Coincidence*** *of a string* $\boldsymbol{s} = s_1 s_2 ... s_t$ *is given by:*

$$\mathrm{IoC}(\boldsymbol{s}) = \frac{\Sigma_{i=1}^{26} N_i(\boldsymbol{s})(N_i(\boldsymbol{s}) - 1)}{t(t-1)},$$

where $N_i(\boldsymbol{s})$ *is the number of times the* i*th letter of the alphabet appears in* $\boldsymbol{s}$.

There is a simple interpretation of this statistic.

Lemma 1.22. $\mathrm{IoC}(\boldsymbol{s})$ *is the probability of picking two symbols uniformly at random from* $\boldsymbol{s}$ *and observing the same letter.*

Proof. The total number of ways of picking two symbols from $\mathbf{s}$ is $\binom{t}{2} = \frac{t(t-1)}{2}$. The number of ways of picking two symbols that are the ith letter of the alphabet is $\binom{N_i(\mathbf{s})}{2} = \frac{N_i(\mathbf{s})(N_i(\mathbf{s})-1)}{2}$ (since there are $N_i(\mathbf{s})$ possibilities to pick from first and then $N_i(\mathbf{s}) - 1$ left to pick from second).

It follows that the probability of picking two symbols uniformly at random from $\mathbf{s}$ and observing the same letter is:

$$\frac{\Sigma_{i=1}^{26} \frac{N_i(\mathbf{s})(N_i(\mathbf{s})-1)}{2}}{\frac{t(t-1)}{2}} = \mathrm{IoC}(\mathbf{s}).$$

□

Example 1.23. *The string*

$$\boldsymbol{s}_1 = \texttt{ZLXRHRRHWLOEHDWEOKLILWVLHPHQBYNWHWFJULRXX}$$

from our running example has Index of Coincidence:

$$\text{IoC}(\boldsymbol{s}_1) = \frac{6(5) + 6(5) + 5(4) + 4(3) + ...}{41 \cdot 40} \approx 0.062,$$

since there are 6 *instances of the letters* `L` *and* `H`*,* 5 *instances of the letter* `W`*,* 4 *instances of the letter* `R` *etc.*

Now suppose that we are given a string $\mathbf{s}$ and are asked to figure out if it came from a monoalphabetic substitution cipher or whether it is generated by sampling from the alphabet uniformly at random. We can use the Index of Coincidence to help us decide this!

Theorem 1.24. *Suppose that $\boldsymbol{s} = s_1 s_2 ... s_t$ is a string of text.*

- *If $\boldsymbol{s}$ was generated by sampling from the alphabet uniformly at random then*

$$\text{IoC}(\boldsymbol{s}) \approx \frac{1}{26} \approx 0.0385.$$

- *If $\boldsymbol{s}$ was generated by a monoalphabetic substitution cipher of English text then*

$$\text{IoC}(\boldsymbol{s}) \approx 0.0656.$$

Proof. (Sketch)

- If the string is generated uniformly then the expect number of occurrences of the ith letter of the alphabet is $\frac{t}{26}$ (since each letter is equally likely to appear).

  It follows that for t large enough:

$$\text{IoC}(\mathbf{s}) \approx \frac{\sum_{i=1}^{26} \frac{t}{26}(\frac{t}{26} - 1)}{t(t-1)} = \frac{1}{26^2} \cdot \frac{\sum_{i=1}^{26} t(t-26)}{t(t-1)} \approx \frac{1}{26^2} \cdot \frac{\sum_{i=1}^{26} t(t-1)}{t(t-1)} = \frac{26}{26^2} = \frac{1}{26}.$$

- First note that if we apply a substitution key $\sigma \in S_{26}$ to $\mathbf{s}$ then the number of occurrences of the ith letter in $\mathbf{s}$ is equal to the number of occurrences of the $\sigma(i)$th letter in $e_\sigma(\mathbf{s})$, so that $N_i(\mathbf{s}) = N_{\sigma(i)}(e_\sigma(\mathbf{s}))$ for each $1 \leq i \leq 26$. It follows that $\text{IoC}(\mathbf{s}) = \text{IoC}(e_\sigma(\mathbf{s}))$ (the terms in the sum are simply permuted, so give the same total).

  We now only need to understand the Index of Coincidence of a piece of English text (since applying any d_σ to $\mathbf{s}$ leaves the value invariant). In a string of English text of length t we expect there to be $N_i \approx \frac{p_i}{100} t$ occurrences of the letter i, where p_i is the number found in row i of the table in Appendix 8.1.

  It follows that for t large enough:

$$\begin{aligned}\text{IoC}(\mathbf{s}) \approx \frac{\sum_{i=1}^{26} \frac{p_i}{100} t(\frac{p_i}{100} t - 1)}{t(t-1)} &= \frac{1}{100^2} \cdot \frac{\sum_{i=1}^{26} p_i^2 t(t-100)}{t(t-1)} \approx \frac{1}{100^2} \cdot \frac{\sum_{i=1}^{26} p_i^2 t(t-1)}{t(t-1)} \\ &= \frac{\sum_{i=1}^{26} p_i^2}{100^2} \approx 0.0656.\end{aligned}$$

□

We can now use the Index of Coincidence as a statistical tool to decide the key length. When we test the correct key length we should get a bunch of partial ciphertext that each have Index of Coincidence close to 0.0656. For incorrect key lengths we will instead observe values close to 0.0385.

Example 1.25. *If we run the pseudoalgorithm on our running example ciphertext for key lengths* $4 \leq n \leq 11$ *we get the following table:*

Key length	IoC values for partial ciphertexts
4	$0.034, 0.042, 0.039, 0.035$
5	$0.038, 0.039, 0.043, 0.027, 0.036$
6	$0.038, 0.038, 0.039, 0.038, 0.032, 0.033$
7	$0.062, 0.057, 0.065, 0.059, 0.060, 0.064, 0.064$
8	$0.037, 0.029, 0.038, 0.030, 0.034, 0.057, 0.040, 0.039$
9	$0.032, 0.036, 0.028, 0.030, 0.026, 0.032, 0.045, 0.047, 0.056$

We see that in most rows the IoC values are mostly low (close to 0.0385*) and so the corresponding partial ciphertexts* $\boldsymbol{s}_i$ *are likely to be "random".*

However, in the row corresponding to key length $n = 7$ *we see that the values are much higher (close to* 0.0656*) and so the corresponding partial ciphertexts* $\boldsymbol{s}_i$ *are likely to be monoalphabetic substitution ciphers.*

Based on this evidence, it looks likely that the key length is $n = 7$*, agreeing with the outcome of the Kasiski test that we did.*

1.6 The Mutual Index of Coincidence

So far, we have two good ways of determining the key size n (Kasiski test and Index of Coincidence), but we are still faced with the problem of decrypting the partial ciphertexts $\mathbf{s}_1, \mathbf{s}_2, ..., \mathbf{s}_n$.

We know that each $\mathbf{s}_i$ has been encrypted by the corresponding key shift k_i, which is a special case of a monoalphabetic substitution cipher...but what do we do beyond this?

The Index of Coincidence told us how to recognise a substitution ciphertext from a uniformly random one. Maybe we can find a statistical tool to tell us when two strings are encrypted using the **same** substitution key?

Definition 1.26. *The* ***Mutual Index of Coincidence*** *of two strings* $\boldsymbol{s} = s_1 s_2 ... s_{t_1}$ *and* $\boldsymbol{r} = r_1 r_2 ... r_{t_2}$ *is given by:*

$$\text{MutIoC}(\boldsymbol{s}, \boldsymbol{r}) = \frac{\Sigma_{i=1}^{26} N_i(\boldsymbol{s}) N_i(\boldsymbol{r})}{t_1 t_2}.$$

As with the IoC, there is a simple interpretation of this statistic.

Lemma 1.27. MutIoC($\boldsymbol{s}, \boldsymbol{r}$) *is the probability of picking two symbols, one uniformly at random from* $\boldsymbol{s}$ *and one uniformly random from* $\boldsymbol{r}$*, and observing the same letter.*

Proof. The total number of ways of picking a pair of symbols, one from $\mathbf{s}$ and one from $\mathbf{r}$, is $t_1 t_2$. The number of ways of picking the ith letter of the alphabet from $\mathbf{s}$ is $N_i(\mathbf{s})$. Similarly, the number of ways of picking the ith letter of the alphabet from $\mathbf{r}$ is $N_i(\mathbf{r})$.

It follows that the probability of picking the pair of symbols and observing the same letter is:

$$\frac{\Sigma_{i=1}^{26} N_i(\mathbf{s}) N_i(\mathbf{r})}{t_1 t_2} = \text{MutIoC}(\mathbf{s}, \mathbf{r}).$$

□

Exercise 1.28. *Show that IoC*($\boldsymbol{s}$) $\approx$ *MutIoC*($\boldsymbol{s}, \boldsymbol{s}$) *for large enough* $t_1 = t_2 = t$ *(the difference being that with* MutIoC *we are allowed to pick the* ***same*** *symbol from* $\boldsymbol{s}$ *twice. This was not the case with* IoC*).*

As promised, we are now ready to see why the Mutual Index of Coincidence is good at telling apart substitution ciphers that have been encrypted using the same key.

Theorem 1.29. *Suppose that* $\boldsymbol{s} = s_1 s_2 ... s_{t_1}$ *and* $\boldsymbol{r} = r_1 r_2 ... r_{t_2}$ *are encrypted strings of English text, using the same monoalphabetic substitution key. Then*

$$\text{MutIoC}(\boldsymbol{s}, \boldsymbol{r}) \approx 0.0656.$$

Proof. (Sketch) First note that if we apply a substitution key $\sigma \in S_{26}$ to $\mathbf{s}$ then the number of occurrences of the ith letter in $\mathbf{s}$ is equal to the number of occurrences of the $\sigma(i)$th letter in $e_\sigma(\mathbf{s})$, so that $N_i(\mathbf{s}) = N_{\sigma(i)}(e_\sigma(\mathbf{s}))$ for each $1 \leq i \leq 26$. Similarly $N_i(\mathbf{r}) = N_{\sigma(i)}(e_\sigma(\mathbf{r}))$. It follows that $\text{MutIoC}(\mathbf{s}, \mathbf{r}) = \text{MutIoC}(e_\sigma(\mathbf{s}), e_\sigma(\mathbf{r}))$ (the terms in the sum are simply permuted, so give the same total).

We now only need to understand the Mutual Index of Coincidence for a pair of English texts (since applying d_σ to both $\mathbf{s}$ and $\mathbf{r}$ leaves the value invariant). In a string of English text of length t we expect there to be $N_i \approx \frac{p_i}{100} t$ occurrences of the letter i, where p_i is the number found in row i of the table in Appendix 8.1.

It follows that for t_1 and t_2 large enough:

$$\text{MutIoC}(\mathbf{s}) \approx \frac{\sum_{i=1}^{26} \frac{p_i}{100} t_1 \cdot \frac{p_i}{100} t_2}{t_1 t_2} = \frac{\sum_{i=1}^{26} p_i^2}{100^2} \approx 0.0656.$$

□

It's a bit harder to pin down what happens if the two strings are encrypted using different keys $\sigma_1, \sigma_2 \in S_{26}$, but in practice you usually find that this gives

$$\text{MutIoC}(\mathbf{s}, \mathbf{r}) \approx \frac{\sum_{i=1}^{26} p_{\sigma_1(i)} p_{\sigma_2(i)}}{100^2} \approx 0.0385.$$

(As we keep saying, it's not an exact science. Some substitution keys won't change the MutIoC value much, but most will).

Exercise 1.30. *Check that this really is the case with a few simple examples.*

We are now ready to launch a full scale attack on the Vigenère cipher.

Assume that we have a Vigenère ciphertext $\mathbf{c}$ and that we have done tests to "determine" the key size n (i.e. Kasiski test or IoC). We split $\mathbf{c}$ into its partial ciphertexts $\mathbf{s}_1, \mathbf{s}_2, ..., \mathbf{s}_n$. Each $\mathbf{s}_i$ is encrypted using a Caesar shift with key k_i (so is a monoalphabetic substitution).

As discussed before we need to determine the shifts k_i. The key idea is to notice that if we shift $\mathbf{s}_j$ by $\beta_{i,j} \equiv k_i - k_j \bmod 26$ for each $1 \leq i \leq n$ then we get a string $e_{\beta_{i,j}}(\mathbf{s}_j)$ that is encrypted with a Caesar shift of k_i, which is the **same** shift that $\mathbf{s}_i$ is encrypted with. It follows that $\mathrm{MutIoC}(\mathbf{s}_i, e_{\beta_{i,j}}(\mathbf{s}_j))$ should be close to 0.0656 (since both are encrypted using the **same** substitution key).

This suggests the following plan of attack:

Pseudoalgorithm

1. For each $1 \leq i < j \leq n$ we compute the list $\{\mathrm{MutIoC}(\mathbf{s}_i, e_\beta(\mathbf{s}_j)) \,|\, 0 \leq \beta \leq 25\}$.
2. We look through these lists and find the values of β that give MutIoC values close to 0.0656, giving us guesses for (some of) the values $\beta_{i,j}$.
3. We solve the system of linear congruences $\beta_{i,j} \equiv k_i - k_j \bmod 26$ and hopefully get a small set of solutions. If not, we go back to the lists and add more equations.
4. Finally, once a small dimensional set of solutions is found, we brute force and look for a decryption that gives a meaningful plaintext.

Again, this is not an exact science, the above pseudoalgorithm can fail. If we make a wrong guess for one of the $\beta_{i,j}$ then the set of equations could have **no** solutions. Some amount of flexibility is often required when making these guesses.

Example 1.31. *Let's return to our running example with key length $n = 7$ and partial ciphertexts:*

$$\boldsymbol{s}_1 = \texttt{ZLXRHRRHWLOEHDWEOKLILWVLHPHQBYNWHWFJULRXX}$$
$$\boldsymbol{s}_2 = \texttt{PAZJZEZZITLWBOAMQBVUZPJPVMTIIMIQUPTIQJKTA}$$
$$\boldsymbol{s}_3 = \texttt{GJPGQPYVVNDFJGJIHJPAGCQCKFKHVQVVCCQPMGTVN}$$
$$\boldsymbol{s}_4 = \texttt{DKYDYVRRSLIIOCYSOOSVMDBFXKOBDMXGBDMDOQIKI}$$
$$\boldsymbol{s}_5 = \texttt{LPYLJMIESIEIWQARAFRMSMRIJRZMAPMEOXOSEEWR}$$
$$\boldsymbol{s}_6 = \texttt{RYGRZSGGAUAYRHGZVRTSEJNOBQRQRBTFRUOFAAVR}$$
$$\boldsymbol{s}_7 = \texttt{JLLZQWZKOWQKHKGQLYGWVSKWJTGSDUZJVWWLVLEG}$$

After tedious computation, we find the table of values $\mathrm{MutIoC}(\boldsymbol{s}_i, e_\beta(\boldsymbol{s}_j))$ *given in Appendix 8.2.*

Looking at the table, we observe quite a few values that are close to 0.0656*:*

i	j	β	$\mathrm{MutIoC}(\boldsymbol{s}_i, e_\beta(\boldsymbol{s}_j))$
1	3	1	0.067
1	4	19	0.071
1	6	16	0.071
2	3	6	0.060
3	4	18	0.073
3	5	24	0.067
3	6	15	0.074
3	7	10	0.069
4	6	23	0.066
4	7	18	0.071
6	7	21	0.069

This leads to the system of linear congruences:

$$\begin{aligned}
k_1 - k_3 &\equiv 1 \bmod 26 & k_3 - k_6 &\equiv 15 \bmod 26 \\
k_1 - k_4 &\equiv 19 \bmod 26 & k_3 - k_7 &\equiv 10 \bmod 26 \\
k_1 - k_6 &\equiv 16 \bmod 26 & k_4 - k_6 &\equiv 23 \bmod 26 \\
k_2 - k_3 &\equiv 6 \bmod 26 & k_4 - k_7 &\equiv 18 \bmod 26 \\
k_3 - k_4 &\equiv 18 \bmod 26 & k_6 - k_7 &\equiv 21 \bmod 26 \\
k_3 - k_5 &\equiv 24 \bmod 26
\end{aligned}$$

The general solution to these equations is given by

$$(\lambda, \lambda - 21, \lambda - 1, \lambda - 19, \lambda - 25, \lambda - 16, \lambda - 11) \qquad \lambda \in \mathbb{Z}/26\mathbb{Z}.$$

Trying each of the 26 *possible values of* λ *and decrypting gives the following:*

λ	Keyword	Decrypted Text
0	AFZHBKP	ZKHWKHULVKDOOWXUQRXWWREHWKHKHURRIPBRZQOLIH...
1	BGAICLQ	YJGVJGTKUJCNNVWTPQWVVQDGVJGJGTQQHOAQYPNKHG...
2	CHBJDMR	XIFUIFSJTIBMMUVSOPVUUPCFUIFIFSPPGNZPXOMJGF...
3	DICKENS	WHETHERISHALLTURNOUTTOBETHEHEROOFMYOWNLIFE...
4	EJDLFOT	VGDSGDQHRGZKKSTQMNTSSNADSGDGDQNNELXNVMKHED...
⋮	⋮	⋮

We immediately recognise that $\lambda = 3$*, giving keyword* DICKENS*, and that the plaintext is the opening passage of David Copperfield:*

WHETHER I SHALL TURN OUT TO BE THE HERO OF MY OWN LIFE OR WHETHER THAT STATION WILL BE HELD BY ANYBODY ELSE THESE PAGES MUST SHOW TO BEGIN MY LIFE WITH THE BEGINNING OF MY LIFE I RECORD THAT I WAS BORN AS I HAVE BEEN INFORMED AND BELIEVE ON A FRIDAY AT TWELVE OCLOCK AT NIGHT IT WAS REMARKED THAT THE CLOCK BEGAN TO STRIKE AND I BEGAN TO CRY SIMULTANEOUSLY

We've broken the Vigenère cipher!

2 Mechanical Cryptography: An industrial revolution in security

The ciphers discussed so far are all examples of historical ciphers, dating from Ancient Rome through to the early 1900's. This was by no means a complete list...there were many other ciphers in use during this time. However, all suffered from similar issues:

1. Encryption/decryption was often a lengthy process, since it had to be done by hand.
2. Because of this, ciphers had to be simple enough in design so that a human could understand and use them.
3. This meant that most ciphers had underlying weaknesses due to underlying language patterns. Statistical attacks rendered these ciphers insecure for large message lengths.

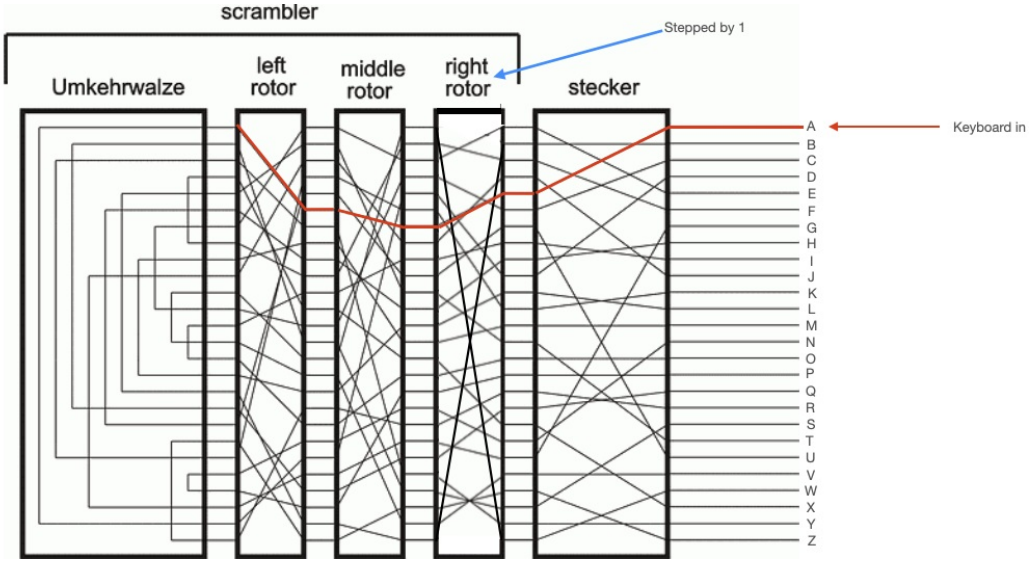
Around the turn of the 20th Century, the Second Industrial Revolution began...bringing with it an age of mechanical discovery. We had now started to harvest electricity and use it to build machines that could perform complicated tasks way more efficiently than humans. Around this time, cryptographers wondered whether machines could be used for encryption/decryption, in such a way that would solve all of the above issues.

In this section we'll study one of the most famous examples of such a machine, the Enigma machine, and learn about one of the most famous feats in classical cryptography, Turing's attack on the Enigma cipher.

2.1 The Enigma cipher

The Enigma machine is perhaps one of the most famous cipher machines in history. Designed by German engineer Arthur Scherbius in 1918, the machine was designed to provide widespread security for commercial, diplomatic and military communications. However, the most well-known use of the Enigma machine was by the Nazi's during World War II.

Let's take a look at the machine in detail. In this course we'll study a slightly simplified version, but the real machine wasn't too different from this.



To use the machine, you would:

- First set up the machine according to a secret choice of machine settings (this is your key).
- Encrypt by typing the plaintext message using the keyboard and recording the corresponding ciphertext letters that light up on the display.
- Decrypt by setting up the machine using the identical key settings, typing the ciphertext message using the keyboard and recording the corresponding plaintext letters that light up on the display.

This is much easier than the previous ciphers we have seen...you didn't even need to know how the machine worked to encrypt/decrypt!

How does the machine work though? Let's open the hood and have a look. The inside of the machine consisted of many electrical wires. Each time a button is pressed on the keyboard it completes a circuit, hence lighting up the corresponding ciphertext letter. The ciphery goodness happens in the middle.

Step 1: The Plugboard (Stecker)

When a letter is pressed on the keyboard, an electrical signal is sent through the plugboard. This consists of a certain number of plugs connecting distinct pairs of letters. This gives a simple substitution cipher that swaps (some) pairs of letters.

How many ways can this be done?

Lemma 2.1. *If m plugs are used then there are $\binom{26}{2m}(2m-1)(2m-3)...(3)(1)$ possible plugboard settings.*

Proof. Choosing m pairs of letters to be swapped is equivalent to first choosing the set of $2m$ letters taking part in the swaps and then choosing how to pair these letters up.

There are $\binom{26}{2m}$ ways to choose $2m$ letters from 26. Fix such a choice.

Pick one of the $2m$ letters. Then there are $2m-1$ choices of letters to pair this letter with. Now choose one of the remaining $2m-2$ letters. There are $2m-3$ choices of letters to pair this letter with. Continuing on we see that there are $(2m-1)(2m-3)...(3)(1)$ ways to form the pairings.

It follows that the total number of plugboard settings is then $\binom{26}{2m}(2m-1)(2m-3)...(3)(1)$. □

Proposition 2.2. *There are a total of* 532985208200576 *plugboard settings.*

Proof. We can use up to 13 plugs and so the total number of plugboard settings is:

$$1+\sum_{1\leq m\leq 13}\binom{26}{2m}(2m-1)(2m-3)...(3)(1)=...=532985208200576.$$

□

In practice, the Germans used exactly 10 plugs, giving a smaller total number of plugboard settings:

$$\binom{26}{20}(19)(17)...(3)(1)=150738274937250.$$

Step 2: The Rotors

So far, all we have done is a simple substitution of letters. However, after the signal has left the plugboard it enters the rotors. These are three "wheels" that have internal wirings that **permute** the letters `A-Z`. The signal passes through the three rotors in turn.

Wait a minute...isn't this just doing yet another bunch of substitution ciphers? Yes it is, but the interesting thing about the rotors is that they **rotate**! After each key is pressed, the right rotor rotates clockwise by one step...hence changing the overall permutation (the internal wiring of the rotor is unchanged). After 26 rotations of the right rotor, the middle rotor rotates one step clockwise. After 26 rotations of the middle rotor, the left rotor rotates one step clockwise.

Proposition 2.3. *The number of initial rotor wirings is:*

$$26!^3=65592937459144468297405473968303761468794234820105359750856704000000000000000000$$

Proof. Since there are 26! possible substitution keys and three independent rotors, there are $26!^3$ possible initial rotor wirings. □

There are a huge number of rotor choices and settings, but in practice the Germans only used **three** of **five** fixed rotors (called I,II,III,IV,V). This leads to a much smaller number of different initial rotor settings:

$$\binom{5}{3}\cdot 6\cdot 26^3=1054560$$

since we must choose the three rotors, their ordering and their starting position (we can rotate them to set up different initial wirings).

Step 3: The Reflector (Umkehrwalze)

After making it through the plugboard and the rotors, the signal is then sent through a reflector. Similar to the plugboard, the internal wiring pairs up **all** of the letters `A-Z`...giving another substitution key.

Proposition 2.4. *There are a total of* 7905853580625 *reflector settings.*

Proof. This is equivalent to the number of plugboard settings with exactly $m = 13$ plugs, i.e.

$$\binom{26}{26}(25)(23)...(3)(1) = 7905853580625.$$

□

Again, there are a huge number of reflectors in general but in practice the Germans only used one of **two** fixed reflectors at any given time (called B and C).

Step 4: The final surprise

After all of this took place, the signal gets sent backwards through the machine. This adds another layer of complexity to the whole thing!

The other reason for doing this (and for including the reflector) is to make encryption/decryption a symmetric process, so that the machine could be used for both without needing different settings.

Now we can ask the big question...how many Enigma keys are there in total?

Corollary 2.5. *The total number of Enigma keys is*

$$\begin{aligned}&276389158444019800290882309785572710383834311090283282\\&283180099646935619811363530997760000000000000000000000.\end{aligned}$$

In practice, the Germans could choose one from the following number of Enigma keys

$$3179251104356527200 00$$

Proof. These numbers are simply the products of the corresponding numbers of keys from steps 1-3. □

Both of these numbers are astronomical and are much bigger than anything we saw before! The Germans chose to communicate their keys via **codebooks**:

Geheim! **Sonder - Maschinenschlüssel BGT**

Datum	Walzenlage	Ringstellung	Steckerverbindungen	Grundstellung
31.	IV II I	F T R	HR AT DW SK UY DF GV LJ BO MX	vyj
30.	III V II	Y V P	OR KI JV GT ZK MU BF YC DS GP	cqr
29.	V IV I	O H R	UX JC Pb bK TA ED ST DS LU FI	vhf

Each day, a new set of machine settings would be used. This made it incredibly difficult for adversaries to keep up to date with the German advances...to be able to read the latest messages any attack essentially needed to work within a day!

How should we think of the Enigma cipher intuitively? Well, the Vigenère cipher extended the Caesar cipher to allow for variable shifts, i.e. $e_{\mathbf{k}}(m_i) \equiv m_i + k_{i \bmod n} \bmod 26$. The Enigma cipher extends the monoalphabetic substitution cipher to allow variable permutations. Before the ith key press the machine is set up to give a substitution key $\sigma_i \in S_{26}$, so that encryption is simply:

$$\begin{aligned} e_{\sigma_i} : \{0,1,...,25\} &\longrightarrow \{0,1,...,25\} \\ m_i &\longmapsto c_i = \sigma_i(m_i) \\ d_{\sigma_i} : \{0,1,...,25\} &\longrightarrow \{0,1,...,25\} \\ c_i &\longmapsto m_i = \sigma_i^{-1}(c_i). \end{aligned}$$

Granted, σ_i is built in a very complicated way...but it's still just a permutation depending on i.

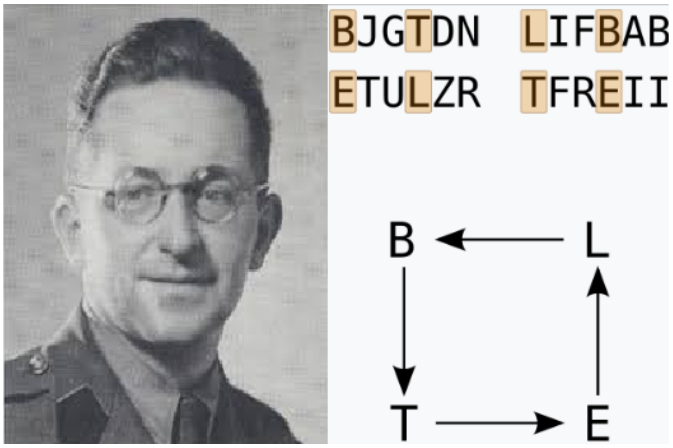
2.2 Attacking Enigma

Given the complicated nature of the Enigma cipher, we might believe that it's completely secure. Surprisingly, it isn't...although it isn't completely insecure! (The common misconception is that Enigma is "completely broken"... but this isn't true, there are still encrypted Enigma messages that we have failed to decipher!).

In this section we'll see roughly how a **known plaintext attack** works. The full story would easily take a full semester of lectures, since this was the product of many years of research by many different groups of mathematicians/cryptographers.

We will make the assumption that we are dealing with the standard German usage of the Enigma machine...so that there are exactly 10 plugs in the plugboard, a choice of 3 from 5 rotors of fixed wiring (that we know) and a choice of two reflectors of fixed wiring (that we know).

Remark 2.6. *We take it for granted that we know the fixed rotor/reflector wirings used by the Germans, but these had to be figured out the hard way. Polish mathematician* ***Marian Rejewski*** *was able to do this using intel from the French...not by using statistical tools but by using results in* ***group theory****. This was the dawn of the usefulness of* ***Pure Mathematics*** *in Cryptography.*



Later he was joined by other Polish cryptographers to launch a full scale attack on Enigma. Rejewski's achievement of understanding the internal wirings used by the Germans is often described as one of the major feats of 20th Century Cryptography.

First, let's model the Enigma machine mathematically. Before the ith key press the machine will perform a permutation $\sigma_i \in S_{26}$. Let's look at σ_i in more detail.

Recall that when a keyboard button is pressed, the signal first travels through the plugboard. Let $P \in S_{26}$ be the fixed permutation corresponding to the plugs inserted.

The signal then goes through three rotors, which induce permutations $R_{1,i}, R_{2,i}, R_{3,i} \in S_{26}$ that depend on i.

After the rotors the signal goes through the reflector, which induces yet another fixed permutation $S \in S_{26}$.

Finally, the signal travels backwards through the rotors and the plugboard. It follows that the permutations we see at this stage are the **inverses** of the permutations going forward.

So all in all:

$$\sigma_i = P^{-1}R_{1,i}^{-1}R_{2,i}^{-1}R_{3,i}^{-1}SR_{3,i}R_{2,i}R_{1,i}P.$$

We've seen things like this in group theory before!

Definition 2.7. *Let G be a group and $g, h \in G$. The **conjugate** of g by h is $h^{-1}gh \in G$.*

The following result about conjugation in permutation groups has been referred to as "the theorem that won World War II".

Theorem 2.8. *Any conjugate of $g \in S_n$ has the same cycle type as g.*

Proof. See the exercise sheet. □

Example 2.9. *Let $g = (14)(235) \in S_5$ be a permutation of cycle type $(2, 3)$ and $h = (12345) \in S_5$. Then:*

$$h^{-1}gh = (15432)(14)(235)(12345) = (124)(35) \in S_5$$

is also a permutation of cycle type $(2, 3)$

Why is this harmless theorem useful for attacking the Enigma cipher?

Corollary 2.10. *The Enigma cipher cannot encrypt a letter to itself.*

Proof. Note that S swaps **all** of the letters `A-Z` in pairs, and so the cycle type of $S \in S_{26}$ is $(2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2)$. It then follows that $\sigma_i \in S_{26}$ also has this cycle type, since it's visibly a conjugate of S.

No such permutation can have a fixed point, and so the substitution key given by σ_i cannot send a letter to itself. □

The above fact proved to be a significant flaw in the Enigma cipher. The reason is that it helps us to locate the **position** of known plaintext.

Example 2.11. *Suppose that we intercept the following ciphertext encrypted with the Enigma cipher:*

XMRPTUBWMZQLFPYH...

If we know that the word CRYPTOGRAPHY *appears in the ciphertext then we can narrow down the potential positions by ignoring positions where letters would be encrypted to themselves:*

```
X M R P T U B G M Z Q L F P Y H . . .
C R Y P T O G R A P H Y
  C R Y P T O G R A P H Y
    C R Y P T O G R A P H Y
      C R Y P T O G R A P H Y
        C R Y P T O G R A P H Y
          C R Y P T O G R A P H Y
```

Once you have some known plaintext and have done the above to figure out where it appears in the ciphertext, it's time for stage two of the attack.

Since the plugboard is the part of the machine with the most possibilities (under the German setup), we try to attack this and instead model σ_i as:

$$\sigma_i = P^{-1}Q_iP,$$

with $Q_i = R_{1,i}^{-1}R_{2,i}^{-1}R_{3,i}^{-1}SR_{3,i}R_{2,i}R_{1,i}$.

The idea is to use the known plaintext/ciphertext pairs and look for "loops". This gives us information about the Q_i that we can hopefully use to narrow down the possibilities.

Example 2.12. *Consider the following plaintext/ciphertext pairs:*

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

We find that the permutations P and Q_i must satisfy the following equations:

$$(P^{-1}Q_9P)(\mathtt{A}) = \mathtt{M}$$
$$(P^{-1}Q_7P)(\mathtt{M}) = \mathtt{E}$$
$$(P^{-1}Q_{14}P)(\mathtt{E}) = \mathtt{A}$$

This implies that $(P^{-1}Q_7Q_9Q_{14}P)(\mathtt{E}) = \mathtt{E}$, since:

$$(P^{-1}Q_7Q_9Q_{14}P) = (P^{-1}Q_7P^{-1}Q_9PP^{-1}Q_{14}P) = (P^{-1}Q_7P)(P^{-1}Q_9P)(P^{-1}Q_{14}P).$$

Rearranging gives $(Q_7Q_9Q_{14})(P(\mathtt{E})) = P(\mathtt{E})$. We learn that whatever the letter $P(\mathtt{E})$ is it must be fixed by the permutation $Q_7Q_9Q_{14}$.

Suppose that we take a guess at the initial rotor and reflector settings. Then we would be able to compute the permutation $Q_7Q_9Q_{14}$, since each of the Q_i are related to these initial configurations by "rotation". If this has no fixed points then we've proved that our guess was impossible!

The natural question now is whether we expect to make progress using the above equation. Unfortunately, the answer is no if we model the permutation $Q_7Q_9Q_{14}$ as uniformly random.

Lemma 2.13. *Let $\sigma \in S_n$ be chosen uniformly random. The expected number of fixed points is 1.*

Proof. Let X be the random variable on S_n measuring the number of fixed points. Then

$$X = X_1 + X_2 + ... + X_n,$$

where

$$X_i = \begin{cases} 1 & \text{if } i \text{ is fixed} \\ 0 & \text{if } i \text{ isn't fixed} \end{cases}.$$

It then follows that:

$$\mathbb{E}[X] = \mathbb{E}[X_1 + X_2 + ... + X_n] = \sum_{i=1}^{n} \mathbb{E}[X_i].$$

If σ is picked uniformly at random then the probability that it fixes i is given by $\frac{(n-1)!}{n!} = \frac{1}{n}$, and so

$$\mathbb{E}[X_i] = 0 \cdot \frac{n-1}{n} + 1 \cdot \frac{1}{n} = \frac{1}{n}.$$

Then the claim follows since

$$\mathbb{E}[X] = \sum_{i=1}^{n} \mathbb{E}[X_i] = n \cdot \mathbb{E}[X_1] = n \cdot \frac{1}{n} = 1.$$

□

However, if we can find **another** loop containing E then we can make progress!

Example 2.14. *Going back to the example, we find that*

$$(P^{-1}Q_4P)(\mathtt{R}) = \mathtt{E}$$
$$(P^{-1}Q_{15}P)(\mathtt{R}) = \mathtt{W}$$
$$(P^{-1}Q_8P)(\mathtt{M}) = \mathtt{W}$$
$$(P^{-1}Q_7P)(\mathtt{M}) = \mathtt{E}$$

This implies that $(P^{-1}Q_4Q_{15}^{-1}Q_8Q_7^{-1}P)\mathtt{E} = \mathtt{E}$ and rearranging gives $(Q_4Q_{15}^{-1}Q_8Q_7^{-1})(P(\mathtt{E})) = P(\mathtt{E})$.

This gives us another permutation that has $P(\mathtt{E})$ as a fixed point.

If we model the pair of permutations $Q_7Q_9Q_{14}$ and $Q_4Q_{15}^{-1}Q_8Q_7^{-1}$ as being uniformly random and independent then we expect there to be $\frac{1}{26}$ fixed points (see exercises). Loosely interpreted, this means that out of the 2109120 possible settings for the rotors/reflector, only around $\frac{2109120}{26} = 81120$ are expected to give a solution to both equations.

This has reduced the number of potential rotor/reflector settings by a factor of 26. If we can find other pairs of cycles then we can continue to reduce the number of possibilities by further factors of 26. In fact, since $\log_{26}(2109120) \approx 4.47$, we require only around 4-5 pairs of loops to uniquely determine the settings.

Once the rotors/reflector settings are known, it is relatively easy to figure out the plugboard settings (we learned a bit about it from the equations, and the known plaintext usually gives enough information to deduce most of the rest).

The attack we described above is (a part of) the attack developed by famous mathematician and cryptographer **Alan Turing**. During World War II the Government Code and Cipher School (now called GCHQ) employed many pure mathematicians, including Turing, to work on breaking the Enigma cipher (secretly based at **Bletchley Park** in Milton Keynes). Such a large scale effort was unheard of up to this point in history, and would provide solid evidence that collaborative efforts could prove much more powerful than working in isolation.

Recall that the attack still required us to do some amount of brute force (the $2 \cdot 1054560 = 2109120$ possibilities for the initial rotor settings and the choice of reflector). By today's standards, this brute force would be possible in seconds...but this was highly non-trivial back in the 1940's. One of Turing's crowning achievements was to design and build a spectacular machine, the **Turing Bombe**, that was able to perform the attack described above.



3 Linear Feedback Shift Registers

We saw earlier that the One Time Pad is a provably secure cipher. However, it suffered from some disadvantages in that we needed to generate a long key uniformly at random. This is not only hard to do, but might be quite time consuming.

We wonder whether there is a way to compromise a little and generate a **pseudorandom** key, i.e. a key that is random enough for you to use as a One Time Pad and get away with it. We also wonder whether we can do this without too much effort...maybe requiring only a **small** input to generate our long random output.

Linear Feedback Shift Registers (LFSR's) give a simple way to do both of these. In this course, we'll only consider LFSR's over $\mathbb{F}_2$, the finite field of order 2, for simplicity (there are much more general notions of LFSR over other fields/rings, e.g. $\mathbb{F}_p$ for other primes p and $\mathbb{Z}/N\mathbb{Z}$ for other integers $N \geq 2$).

3.1 The basics

Definition 3.1. *Let $a, b \in \mathbb{F}_2$. Then the* **XOR** *of a and b, denoted $a \oplus b$ is given by the following table:*

$a \backslash b$	0	1
0	0	1
1	1	0

The term **XOR** stands for **Exclusive OR**, since this is almost the logical operation **OR** (but it excludes **BOTH**). Note that this is simply addition mod 2, but we adopt the notation since it's a more natural notation from the point of view of Computer Science.

Definition 3.2. *A* ***Linear Feedback Shift Register of length*** $n \geq 1$ *with* ***taps*** $\boldsymbol{b} = (b_0, b_1, ..., b_{n-1}) \in \mathbb{F}_2^n$ *is a device that stores a state* $\boldsymbol{a_i} \in \mathbb{F}_2^n$ *at each (integer) time step* $i \geq 0$ *and updates over time as follows:*

$$\boldsymbol{a_i} = (a_i, a_{i+1}, ..., a_{i+n-1}) \quad \longmapsto \quad \boldsymbol{a_{i+1}} = (a_{i+1}, a_{i+2}, ..., a_{i+n}),$$

where $a_{i+n} = b_0 a_i \oplus b_1 a_{i+1} \oplus ... \oplus b_{n-1} a_{i+n-1}$.

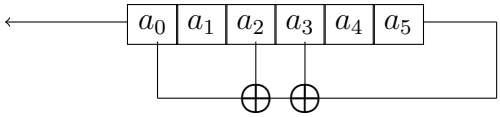
The starting string $\boldsymbol{a}_0 = (a_0, a_1, ..., a_{n-1}) \in \mathbb{F}_2^n$ *is called the* ***initial fill****.*

In other words, at each time step the current state shifts to the left, removing the leftmost entry and updating the rightmost entry with a new bit (depending recursively on the old state).

We'll often draw diagrams to illustrate LFSR's as follows:

Example 3.3. *The LFSR of length $n = 6$ with taps $\boldsymbol{b} = (b_0, b_1, b_2, b_3, b_4, b_5) = (1, 0, 1, 1, 0, 0)$ has update rule:*

$$a_{i+6} = a_i \oplus a_{i+2} \oplus a_{i+3}.$$



If the initial fill is $\boldsymbol{a}_0 = (a_0, a_1, a_2, a_3, a_4, a_5) = (0, 1, 1, 0, 1, 0)$ *then over time the LFSR will store the states:*

$\boldsymbol{a}_0$	0	1	1	0	1	0
$\boldsymbol{a}_1$	1	1	0	1	0	1
$\boldsymbol{a}_2$	1	0	1	0	1	0
$\boldsymbol{a}_3$	0	1	0	1	0	0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

If we run the LFSR forever and at each time step collect the left most bit (the one that gets thrown away), then we recover an infinite string of bits:

$$\mathbf{s} = a_0 a_1 a_2 ... a_n ...$$

which can be used as a One Time Pad key (if we convert our plaintext message into bits using a standard method, such as ASCII (see Appendix 8.3)).

Example 3.4. *The sequence of bits we get from the LFSR in the previous example is:*

$$\boldsymbol{s} = 01101010010001011111101...$$

The plaintext message `HI` *is first encoded into* ASCII *as* 0100100001001001*.*

We then encrypt using the string $\boldsymbol{s}$ *as a One Time Pad (although adding mod* 2*):*

m_i	0	1	0	0	1	0	0	0	0	1	0	0	1	0	0	1
a_i	0	1	1	0	1	0	1	0	0	1	0	0	0	1	0	1
$c_i = m_i \oplus a_i$	0	0	1	0	0	0	1	0	0	0	0	0	1	1	0	0

and receive the ciphertext $\boldsymbol{c} = 0010001000001100$*.*

We could also use the bit string to encrypt pictures made up of black and white pixels (in the same way that we saw in the One Time Pad chapter).

It's now clear why LFSR's are useful in Cryptography, we can generate **long** "random-looking" One Time Pad keys by using **small** initial fills (which must be kept secret). This has the obvious advantage that the key is easier to communicate with someone...they would only need to know the initial fill and the update rule used to be able to generate **s**.

The only downside is that the key isn't **really** random...it was generated using a recursion, so is **deterministic**. LFSR's can only generate pseudorandom strings, and some LFSR's are better than others in this regard!

Example 3.5. *The randomness of the output string* $\boldsymbol{s}$ *of an LFSR can depend heavily on the choice of initial fill.*

Suppose we use the LFSR of length $n = 6$ *from earlier, but now use the initial fill* $\boldsymbol{a}_0 = (0, 0, 0, 0, 0, 0)$. *This generates the string* $\boldsymbol{s} = 0000000000000....$

I think we can all agree that this is not a very random looking string. Not only that but it would be a completely useless One Time Pad key (it performs the trivial encryption!).

Similarly, the initial fill $\boldsymbol{a}_0 = (1, 1, 1, 1, 1, 1)$ *generates the string* $\boldsymbol{s} = 1111111111111...$, *which is also a very predictable sequence and a bad One Time Pad key.*

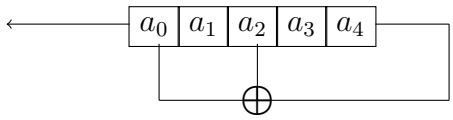
However, as we saw earlier the initial fill $\boldsymbol{a}_0 = (0, 1, 1, 0, 1, 0)$ *led to a more random looking string* $\boldsymbol{s} = 01101010010000101111101....$

Example 3.6. *The randomness of the output string* $\boldsymbol{s}$ *can also depend heavily on the choice of taps.*

Consider the LFSR of length 5 *with taps* $\boldsymbol{b} = (1, 0, 1, 0, 0)$. *This gives update rule*

$$a_{i+5} = a_i \oplus a_{i+2}$$

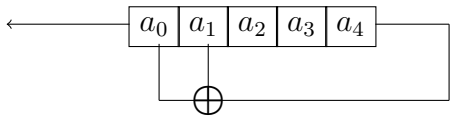
and picture:



It turns out that any choice of initial fill (except $\boldsymbol{a}_0 = (0, 0, 0, 0, 0)$*) gives a random looking string* $\boldsymbol{s}$. *For example, the initial fill* $\boldsymbol{a}_0 = (0, 1, 1, 0, 1)$ *gives:*

$$\boldsymbol{s} = 0110111010100001001 01...$$

However, the LFSR of length 5 *with taps* $\boldsymbol{b} = (1, 1, 0, 0, 0)$ *gives update rule* $a_{i+5} = a_i \oplus a_{i+1}$ *and picture:*



For the same initial fill as above we get the output string

$$\boldsymbol{s} = 01101101101101101101 1...$$

which looks very predictable!

So now we ask the natural question, is it possible to look at an LFSR and **predict** its quality? How "pseudorandom" can its output strings be, based on the taps and the initial fill?

Clearly this is a vague question...how are we **quantifying** pseudorandomness? A good way to measure this is by how often the string **s** repeats.

Definition 3.7. *Consider an LFSR with initial fill* $\boldsymbol{a}_0$*. Then the* ***period*** *of* $\boldsymbol{a}_0$ *is the smallest* $j \geq 1$ *such that* $\boldsymbol{a}_{i+j} = \boldsymbol{a}_i$ *for some* $i \geq 0$ *(i.e. the smallest number of time steps needed to observe a repeat between two states of the LFSR).*

We denote this by $\mathrm{Per}(\boldsymbol{a}_0) = j$*.*

Example 3.8. *In the previous example we considered two LFSR's of length* 5 *and the same initial fill* $\boldsymbol{a}_0 = (0, 1, 1, 0, 1)$*.*

The second clearly satisfies $\mathrm{Per}(\boldsymbol{a}_0) \leq 3$*, since* $\boldsymbol{a}_{i+3} = \boldsymbol{a}_i$ *for every* $i \geq 0$*. Since we see visibly that* $\boldsymbol{a}_{i+1} \neq \boldsymbol{a}_i$ *and* $\boldsymbol{a}_{i+2} \neq \boldsymbol{a}_i$ *for all* $i \geq 0$*, we have that* $\mathrm{Per}(\boldsymbol{a}_0) = 3$*.*

For the first LFSR we note that the computation above didn't reveal any repeats, so we must compute further:

$$\boldsymbol{s} = 01101110101000010010110011111000 1101...$$

A quick count reveals that $\boldsymbol{a}_{31} = \boldsymbol{a}_0$*. This implies that* $\boldsymbol{a}_{i+31} = \boldsymbol{a}_i$ *for every* $i \geq 0$ *(Why?), and so* $\mathrm{Per}(\boldsymbol{a}_0) \leq 31$*. We must have equality since if* $\boldsymbol{a}_{i+j} = \boldsymbol{a}_i$ *for some* $1 \leq j \leq 30$ *and* $i \geq 0$*, then by periodicity we would have this equality for some* $0 \leq i \leq 30$ *(and visibly there are no repeats in the states* $\boldsymbol{a}_0, \boldsymbol{a}_1, ..., \boldsymbol{a}_{30}$*).*

Thus $\mathrm{Per}(\boldsymbol{a}_0) = 31$*. (Later, we'll see why for this LFSR the period is always* 31 *for* ***any*** *non-zero initial fill).*

Ok, so now that we have a way of quantifying the pseudorandomness of an LFSR, two questions remain:

- How do we compute $\mathrm{Per}(\mathbf{a}_0)$? We'll see a partial answer to this question soon!
- Does $\mathrm{Per}(\mathbf{a}_0)$ actually exist? Is it always the case that the output string of an LFSR always repeats from some point onwards?

Proposition 3.9. *An LFSR of length* n *satisfies* $\mathrm{Per}(\boldsymbol{a}_0) \leq 2^n - 1$ *for every initial fill* $\boldsymbol{a}_0$*.*

Proof. Suppose that $\mathbf{a}_0 = (0, 0, ..., 0)$. Then $\mathrm{Per}(\mathbf{a}_0) = 1 \leq 2^n - 1$ (see exercises).

Now suppose that $\mathbf{a}_0 \neq (0, 0, ..., 0)$. If at some time step we have $\mathbf{a}_i = (0, 0, ..., 0)$ then the above tells us that $\mathrm{Per}(\mathbf{a}_0) = 1 \leq 2^n - 1$. So we may assume that $\mathbf{a}_i \neq (0, 0, ..., 0)$ for any $i \geq 0$.

There are $2^n - 1$ elements of $\mathbb{F}_2^n \backslash \{(0, 0, ..., 0)\}$. Now consider the states $\mathbf{a}_0, \mathbf{a}_1, ..., \mathbf{a}_{2^n-1}$ of the LFSR. There are 2^n such states, and so by the Pigeonhole principle we must have that $a_n = a_m$ for some $0 \leq m < n \leq 2^n - 1$. It follows that $\mathrm{Per}(\mathbf{a}_0) \leq n - m \leq 2^n - 1$. □

3.2 The Main Theorem

We've seen that the quality of an LFSR as a tool for generating pseudorandom bit strings is measured by the period of its initial fill. We know that this period satisfies $\mathrm{Per}(\mathbf{a}_0) \leq 2^n - 1$, but we would really like to know its **exact** value! We'll see a partial answer to this question in this subsection.

First we make the simple observation that if the first tap satisfies $b_0 = 0$ (i.e. we never plan to use the first bit stored in the register in any of our computations) then we are really just implementing a **smaller** LFSR (i.e. a truncated one).

More precisely:

Lemma 3.10. *If an LFSR of order n has taps $\boldsymbol{b} = (0, 0, ..., 0, b_k, ..., b_{n-1})$ for some $k \geq 1$ then the output string corresponding to initial fill $\boldsymbol{a}_0 = (a_0, a_1, ..., a_{n-1})$ is*

$$\boldsymbol{s} = a_0 a_1 ... a_{k-1} \boldsymbol{s'},$$

where $\boldsymbol{s'}$ is the output string of the LFSR of order $n-k$ with taps $\boldsymbol{b'} = (b_k, b_{k+1}, ..., b_{n-1})$ and initial fill $\boldsymbol{a}'_0 = (a_k, a_{k+1}, ..., a_{n-1})$.

In particular $Per(\boldsymbol{a}_0) = Per(\boldsymbol{a}'_0)$.

Exercise 3.11. *Prove this rigorously.*

Because of the above fact, we may assume that our LFSR has first tap satisfying $b_0 = 1$. The importance of this will be clear soon.

So what will we do now? Well, we haven't yet used the fact that the word **linear** appears in the acronym LFSR. Consider what happens in the time step $i \mapsto i + 1$ of the LFSR:

$$\begin{aligned}
\mathbf{a}_i = (a_i, a_{i+1}, ..., a_{i+n-1}) \longmapsto \mathbf{a}_{i+1} &= (a_{i+1}, a_{i+2}, ..., a_{i+n}) \\
&= (a_{i+1}, a_{i+2}, ..., b_0 a_i \oplus b_1 a_{i+1} \oplus ... \oplus b_{n-1} a_{i+n-1}) \\
&= (a_i, a_{i+1}, ..., a_{i+n-1}) \begin{pmatrix} 0 & 0 & 0 & ... & 0 & b_0 \\ 1 & 0 & 0 & ... & 0 & b_1 \\ 0 & 1 & 0 & ... & 0 & b_2 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & ... & 1 & b_{n-1} \end{pmatrix} \\
&= \mathbf{a}_i \begin{pmatrix} 0 & 0 & 0 & ... & 0 & b_0 \\ 1 & 0 & 0 & ... & 0 & b_1 \\ 0 & 1 & 0 & ... & 0 & b_2 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & ... & 1 & b_{n-1} \end{pmatrix}.
\end{aligned}$$

This is just matrix multiplication!

Definition 3.12. *The **companion matrix** of an LFSR of order n and taps $\boldsymbol{b} = (b_0, b_1, ..., b_{n-1})$*

is the matrix:

$$B_{\boldsymbol{b}} = \begin{pmatrix} 0 & 0 & 0 & ... & 0 & b_0 \\ 1 & 0 & 0 & ... & 0 & b_1 \\ 0 & 1 & 0 & ... & 0 & b_2 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & ... & 1 & b_{n-1} \end{pmatrix} \in M_n(\mathbb{F}_2).$$

We immediately learn some things about the period of an LFSR.

Proposition 3.13. *Consider an LFSR with taps $\boldsymbol{b}$ and initial fill $\boldsymbol{a}_0$. Suppose $b_0 = 1$.*

1. *We have that* $\mathrm{Per}(\boldsymbol{a}_0) \,|\, \mathrm{ord}(B_{\boldsymbol{b}})$ *(i.e. the smallest $k \in \mathbb{N}$ such that $B_{\boldsymbol{b}}^k = I$).*
2. *We have that* $\mathrm{Per}(\boldsymbol{a}_0)$ *is the smallest j such that $\boldsymbol{a}_0 = \boldsymbol{a}_j$ (i.e. to find the period it's enough to just look for the first repeat of the initial fill $\boldsymbol{a}_0$).*

Proof. 1. Suppose that $k = \mathrm{ord}(B_{\mathbf{b}})$. Then $B_{\mathbf{b}}^k = I$ and so since:

$$\mathbf{a}_{i+k} = \mathbf{a}_{i+k-1}B_{\mathbf{b}} = \mathbf{a}_{i+k-2}B_{\mathbf{b}}^2 = ... = \mathbf{a}_i B_{\mathbf{b}}^k = \mathbf{a}_i I = \mathbf{a}_i,$$

we have that $\mathbf{a}_{i+k} = \mathbf{a}_i$ for any $i \geq 0$. It follows that $\mathrm{Per}(\mathbf{a}_0) \leq k = \mathrm{ord}(B_{\mathbf{b}})$.

To get divisibility, suppose that $\mathrm{Per}(\mathbf{a}_0) = j \leq k$. Choose $i \geq 0$ such that $\mathbf{a}_{i+j} = \mathbf{a}_i$. Writing $k = qj + r$ for some $q \geq 1$ and $0 \leq r < j$ we see that:

$$\mathbf{a}_i = \mathbf{a}_{i+j} = ... = \mathbf{a}_{i+qj}.$$

But we also know that $\mathbf{a}_i = \mathbf{a}_{i+k}$, so that $\mathbf{a}_{i+qj} = \mathbf{a}_{i+k}$. If $r \neq 0$ then this implies that $\mathrm{Per}(\mathbf{a}_0) \leq r < j$, which is a contradiction. Thus $r = 0$, so that $j|k$.

2. Suppose that $i \geq 0$ and $j \geq 1$ are such that $\mathbf{a}_{i+j} = \mathbf{a}_i$. Then $\mathbf{a}_0 B_{\mathbf{b}}^{i+j} = \mathbf{a}_0 B_{\mathbf{b}}^i$. Note that $\det(B_{\mathbf{b}}) = b_0 = 1$, and so $B_{\mathbf{b}} \in \mathrm{GL}_n(\mathbb{F}_2)$ (i.e. is invertible mod 2). Cancelling $B_{\mathbf{b}}^i$ from both sides of the equation gives $\mathbf{a}_0 B_{\mathbf{b}}^j = \mathbf{a}_0$, i.e. $\mathbf{a}_j = \mathbf{a}_0$. It follows that any repetition in the sequence $\mathbf{a}_0, \mathbf{a}_1, \mathbf{a}_2, ...$ descends to a repetition with the starting term $\mathbf{a}_0$, and so the period is the smallest j such that $\mathbf{a}_j = \mathbf{a}_0$.

□

Example 3.14. *Consider the LFSR of length $n = 4$ with taps $\boldsymbol{b} = (1, 0, 1, 1)$. Then:*

$$B_{\mathbf{b}} = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix}.$$

By computing powers of $B_{\mathbf{b}}$ (remembering to work mod 2) we find that the lowest power giving the identity matrix is $B_{\mathbf{b}}^7 = I$. It follows that $\mathrm{Per}(\boldsymbol{a}_0) \,|\, \mathrm{ord}(B_{\mathbf{b}}) = 7$*, so that each initial fill $\boldsymbol{a}_0$ has period* $\mathrm{Per}(\boldsymbol{a}_0) = 1$ *or* 7*. This is much better than the result from earlier, which only told us that* $\mathrm{Per}(\boldsymbol{a}_0) \leq 2^4 - 1 = 15$.

Ok, so we now know quite a bit more about the period of an LFSR...but we still don't know **exactly** what it is!

It turns out that we can give an exact answer in one special case. Before we get to this we need to define one more thing.

Definition 3.15. *The **characteristic polynomial** of the LFSR of order n with taps $\boldsymbol{b} = (b_0, b_1, ..., b_{n-1})$ is the polynomial:*

$$c_{\boldsymbol{b}}(x) = x^n + b_{n-1}x^{n-1} + ... + b_1 x + b_0 \in \mathbb{F}_2[x].$$

The reason for the name is that $c_{\mathbf{b}}(x)$ is also the characteristic polynomial of the matrix $B_{\mathbf{b}}$, i.e. $\det(B_{\mathbf{b}} - xI)$ (check this!).

Theorem 3.16. *Suppose that an LFSR with length $n \geq 1$ and taps $\boldsymbol{b}$ has **irreducible** characteristic polynomial $c_{\boldsymbol{b}}(x)$. Let $j \geq 1$ be the smallest positive integer $j \mid 2^n - 1$ such that $c_{\boldsymbol{b}}(x) \mid x^j + 1 \in \mathbb{F}_2[x]$. Then $\mathrm{Per}(\boldsymbol{a}_0) = j$ for every non-zero initial fill $\boldsymbol{a}_0$.*

Proof. Consider the quotient ring

$$\mathbb{F}_2[x]/\langle c_{\mathbf{b}}(x)\rangle = \{[a_0 + a_1 x + a_2 x^2 + ... + a_{n-1}x^{n-1}] \mid a_i \in \mathbb{F}_2\}.$$

This is a vector space over $\mathbb{F}_2$ with (ordered) basis $[1], [x], [x^2], ..., [x^{n-1}]$.

The multiplication by $[x]$ map:

$$\begin{aligned} \mathbb{F}_2[x]/\langle c_{\mathbf{b}}(x)\rangle &\longrightarrow \mathbb{F}_2[x]/\langle c_{\mathbf{b}}(x)\rangle \\ [f(x)] &\longmapsto [xf(x)] \end{aligned}$$

is a linear map which sends:

$$\begin{aligned} [1] &\longmapsto [x] \\ [x] &\longmapsto [x^2] \\ &\vdots \\ [x^{n-1}] &\longmapsto [x^n] = [b_{n-1}x^{n-1} + ... + b_1 x + b_0] = b_0[1] + b_1[x] + ... + b_{n-1}[x^{n-1}] \end{aligned}$$

and so has matrix:

$$\begin{pmatrix} 0 & 0 & 0 & ... & 0 & b_0 \\ 1 & 0 & 0 & ... & 0 & b_1 \\ 0 & 1 & 0 & ... & 0 & b_2 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & ... & 1 & b_{n-1} \end{pmatrix} = B_{\mathbf{b}}.$$

If we identify

$$\mathbf{a}_0 = (a_0, a_1, a_2, ..., a_{n-1}) \longmapsto [a_0(x)] = [a_0 + a_1 x + a_2 x^2 + ... + a_{n-1}x^{n-1}]$$

it follows that $\mathrm{Per}(\mathbf{a}_0)$ is the same as the period of the multiplication by $[x]$ map with starting value $[a_0(x)]$, i.e. the smallest j such that $[x^j a_0(x)] = [a_0(x)]$.

If $c_{\mathbf{b}}(x)$ is irreducible then it is a fact that $\mathbb{F}_2[x]/\langle c_{\mathbf{b}}(c)\rangle$ is a field (i.e. every non-zero element is invertible).

Using this fact, we see that if $\mathbf{a}_0$ is a non-zero initial fill then $[a_0(x)]$ is also non-zero, and so cancelling tells us that $[x^j a_0(x)] = [a_0(x)]$ if and only if $[x^j] = [1]$. This means that $\mathrm{Per}(\mathbf{a}_0) = \mathrm{ord}(x)$ for any such initial fill. Since the group of non-zero elements of our field has size $2^n - 1$, Lagrange's theorem tells us that $\mathrm{ord}(x) \mid 2^n - 1$.

Finally, we have that:

$$[x^j] = [1] \quad \Leftrightarrow \quad [x^j + 1] = [0] \quad \Leftrightarrow \quad x^j + 1 \in \langle c_{\mathbf{b}}(x)\rangle \quad \Leftrightarrow \quad c_{\mathbf{b}}(x) \mid x^j + 1,$$

as desired. □

Example 3.17. *Consider an LFSR of length $n = 4$ with irreducible characteristic polynomial $c_{\boldsymbol{b}}(x)$. Then the above theorem tells us that every non-zero initial fill $\boldsymbol{a}_0$ has the same period, $\mathrm{Per}(\boldsymbol{a}_0) \mid 2^4 - 1 = 15$. We determine which by finding the smallest divisor $j \mid 15$ such that $c_{\boldsymbol{b}}(x) \mid x^j + 1$.*

Note that the divisors of 15 are $1, 3, 5, 15$ and that we have the following factorisations into irreducibles:

$$x^3 + 1 = (x + 1)(x^2 + x + 1)$$
$$x^5 + 1 = (x + 1)(x^4 + x^3 + x^2 + x + 1)$$

1. *If $\boldsymbol{b} = (1, 1, 1, 1)$ then $c_{\boldsymbol{b}}(x) = x^4 + x^3 + x^2 + x + 1$ can be checked to be irreducible (it doesn't have a root mod 2 and doesn't factor as $(x^2 + x + 1)^2$). Since $c_{\boldsymbol{b}}(x) \mid x^5 + 1$ but $c_{\boldsymbol{b}}(x) \nmid x + 1$, we have that $\mathrm{Per}(\boldsymbol{a}_0) = 5$ for every non-zero initial fill $\boldsymbol{a}_0$.*

   For example, $\boldsymbol{a}_0 = (1, 0, 0, 1)$ gives the string:

   $$\boldsymbol{s} = 100101001010 01...$$

2. *If $\boldsymbol{b} = (1, 1, 0, 0)$ then $c_{\boldsymbol{b}}(x) = x^4 + x + 1$ can be checked to be irreducible (same reasoning). Since $c_{\boldsymbol{b}}(x) \nmid x^j + 1$ for any $j \in \{1, 3, 5\}$, we conclude that $\mathrm{Per}(\boldsymbol{a}_0) = 15$ for any non-zero initial fill $\boldsymbol{a}_0$ (i.e. all non-zero initial fills have maximal period).*

   For example, $\boldsymbol{a}_0 = (1, 0, 0, 1)$ gives the string:

   $$\boldsymbol{s} = 100110101111000 1001...$$

3. *If $\boldsymbol{b} = (1, 0, 1, 0)$ then $\boldsymbol{c_b}(x) = x^4 + x^2 + 1 = (x^2 + x + 1)^2$ is reducible and so we cannot use the theorem in this case.*

   To demonstrate this, note that the initial fill $\boldsymbol{a}_0 = (0, 1, 0, 1)$ gives the string:

   $$01010101...$$

   of period $\mathrm{Per}(\boldsymbol{a}_0) = 2 \nmid 15$. Note also that the initial fill $\boldsymbol{a}_0 = (0, 0, 1, 1)$ gives a string of period $4 \neq 2$ (so that different initial fills can also have different periods!).

4 Public Key Cryptography: Asymmetric security in the digital era

We saw earlier that the invention of computational machines provided a whole host of new capabilities in Cryptography. Such machines could be used to both **implement** and **break** ciphers in ways that we could only dream of before.

As time went on the performance of these machines rapidly increased. They also became more readily available on a commercial scale (almost everyone has access to a computer now!). As such, the **Digital Era** (that we currently live in) has brought with it new challenges:

- In a data-driven society it has become more important than ever to find ways of keeping personal data **secure** (e.g. financial data, healthcare data). This is very much a different viewpoint than the "cloak and dagger" style Cryptography that we've seen previously...in the modern world anyone's personal data can be a target!
- Cryptography has had to keep up with technological advances! The security of a scheme now depends how how good the **current capabilities** are (e.g. can I simply invest a billion pounds into building a good enough supercomputer to break someone's security).

Over the last 50-60 years there have been many new and wonderful ciphers developed to cope with the rise in demand (e.g. DES, 3DES, AES, Blowfish). These new schemes are much more complicated to understand by humans, but are designed to be easily carried out by computers (e.g. iterating intricate sequences of non-linear operations many times).

Alongside these developments, a revolution happened in Cryptography in the 1970's:

- It was discovered that we could use ideas from **Pure Mathematics** to **provide** our security.
- We could also use ideas from **Pure Mathematics** to **communicate** without needing a shared key.

The main idea is to shift focus. Up to this point our ciphers have been **symmetric**, i.e. knowing the encryption key is equivalent to knowing the decryption key. Kerckhoffs' principle told us that the security depended entirely on keeping the key private.

The idea of **Public key cryptography** is to make the situation **asymmetric**, using both public and private information to share keys or communicate.

4.1 Diffie-Hellman Key Exchange

So far, all ciphers we have seen have required you to communicate the key with your friend. If this is intercepted in any way then the security is broken!

Question 4.1. *Is there a way for two people to communicate a* ***shared key*** *without either having to send it to the other?*

The above sounds impossible at first glance...but we are about to see that it's very much possible using Modular Arithmetic.

Recall that for each prime $p \geq 1$ we have the group of **units mod** p:

$$(\mathbb{Z}/p\mathbb{Z})^\times = \{\overline{1}, \overline{2}, ..., \overline{p-1}\}.$$

We'll need the following fact, which we will state without proof.

Theorem 4.2. $(\mathbb{Z}/p\mathbb{Z})^\times$ *is* ***cyclic****, i.e. there exists* $\overline{g} \in (\mathbb{Z}/p\mathbb{Z})^\times$ *such that every* $\overline{h}$ *can be written as* $\overline{h} = \overline{g}^n$ *for some* $n \in \mathbb{Z}$.

Definition 4.3. *An element* $\overline{g} \in (\mathbb{Z}/p\mathbb{Z})^\times$ *satisfying the above property is called a* ***primitive root****.*

Example 4.4. *Show that* $\overline{2} \in (\mathbb{Z}/7\mathbb{Z})^\times$ *is not a primitive root. Show that* $\overline{3} \in (\mathbb{Z}/7\mathbb{Z})^\times$ *is a primitive root.*

Solution. The powers of $\overline{2} \in (\mathbb{Z}/7\mathbb{Z})^\times$ are:

$$\overline{2}^1 = \overline{2}, \qquad \overline{2}^2 = \overline{4}, \qquad \overline{2}^3 = \overline{1}, \qquad \overline{2}^4 = \overline{2}, \qquad \overline{2}^5 = \overline{4}, \qquad \overline{2}^6 = \overline{1}, ...$$

and so $\overline{2} \in (\mathbb{Z}/7\mathbb{Z})^\times$ is not a primitive root (e.g. we cannot write $\overline{3} = \overline{2}^n$).

The powers of $\overline{3} \in (\mathbb{Z}/7\mathbb{Z})^\times$ are:

$$\overline{3}^1 = \overline{3}, \qquad \overline{3}^2 = \overline{2}, \qquad \overline{3}^3 = \overline{6}, \qquad \overline{3}^4 = \overline{4}, \qquad \overline{3}^5 = \overline{5}, \qquad \overline{3}^6 = \overline{1}, ...$$

and so $\overline{3} \in (\mathbb{Z}/7\mathbb{Z})^\times$ is a primitive root. ■

In general, a primitive element is easy to find. For example, heuristics say that $\overline{2} \in (\mathbb{Z}/p\mathbb{Z})^\times$ is a primitive root for roughly 37.4% of primes p (although it is still an open problem to prove that this is true, it is a case of the **Artin Primitive Root Conjecture**).

We're now ready to see the Diffie-Hellman method of generating shared keys.

Diffie-Hellman Key Exchange

1. Alice and Bob (publically) agree on a prime number $p \geq 1$ and a primitive root $\overline{g} \in (\mathbb{Z}/p\mathbb{Z})^\times$.
2. Alice chooses a **private key** $k_A \in \mathbb{Z}$ and sends $\overline{c_A} = \overline{g}^{k_A}$ to Bob.
3. Bob chooses a **private key** $k_B \in \mathbb{Z}$ and sends $\overline{c_B} = \overline{g}^{k_B}$ to Alice.
4. Alice computes $\overline{c} = \overline{c_B}^{k_A}$ and Bob computes $\overline{c} = \overline{c_A}^{k_B}$. This is their shared key.

The above protocol was invented by Whitfield Diffie and Martin Hellman in 1976. Before seeing an example, we show that it does indeed work.

Proposition 4.5. *The above protocol generates the same key for Alice and Bob.*

Proof. This is a simple calculation:

$$\overline{c_B}^{k_A} = (\overline{g}^{k_B})^{k_A} = \overline{g}^{k_B k_A} = \overline{g}^{k_A k_B} = (\overline{g}^{k_A})^{k_B} = \overline{c}_A^{k_B}.$$

□

Example 4.6. *Alice and Bob decide to use Diffie-Hellman Key Exchange with* $p = 89$ *and* $\overline{g} = \overline{3} \in (\mathbb{Z}/89\mathbb{Z})^\times$. *Alice decides to use private key* $k_A = 10$ *and Bob decides to use private key* $k_B = 25$. *What messages do they send to each other and what is their shared key?*

Solution. Alice sends the following to Bob:

$$c_A \equiv g^{k_A} \equiv 3^{10} \equiv (-8)^2 \cdot 9 \equiv 576 \equiv \mathbf{42} \bmod 89.$$

Bob sends the following to Alice:

$$c_B \equiv g^{k_B} \equiv 3^{25} \equiv (-8)^6 \cdot 3 \equiv (-25)^3 \cdot 3 \equiv 625 \cdot 14 \equiv 2 \cdot 14 \equiv \mathbf{28} \bmod 89.$$

The shared key is given by:

$$c \equiv c_A^{k_B} \equiv 42^{25} \equiv 6^{25} \cdot 7^{25} \equiv 20^4 \cdot (-13)^8 \cdot 6 \cdot 7 \equiv 44^2 \cdot (-9)^4 \cdot 6 \cdot 7 \equiv 67 \cdot 64 \cdot 6 \cdot 7 \equiv \mathbf{49} \bmod 89.$$

We check this as follows:

$$c \equiv c_B^{k_A} \equiv 28^{10} \equiv 4^{10} \cdot 7^{10} \equiv (-11)^2 \cdot (-2)^2 \cdot 16 \cdot 49 \equiv 32 \cdot 4 \cdot 16 \cdot 49 \equiv 39 \cdot (-17) \equiv \mathbf{49} \bmod 89.$$

■

Now that we understand how Diffie-Hellman Key Exchange works, we discuss what makes this secure.

Eve the evesdropper is interested in getting their hands on the shared key, so that they can read future messages. Eve knows the prime $p \geq 1$ and the primitive root $\overline{g} \in (\mathbb{Z}/p\mathbb{Z})^\times$, and has intercepted the quantities $\overline{c_A} = \overline{g}^{k_A}$ and $\overline{c_B} = \overline{g}^{k_B}$. In order to calculate the shared key $\overline{c}$, Eve has to solve the following problem.

Diffie-Hellman Problem: Given $\overline{g}^x \in (\mathbb{Z}/p\mathbb{Z})^\times$ and $\overline{g}^y \in (\mathbb{Z}/p\mathbb{Z})^\times$, compute $\overline{g}^{xy} \in (\mathbb{Z}/p\mathbb{Z})^\times$.

Generically, it turns out that this problem is very difficult to solve if p is **large**, unless you know either of the quantities x or y. In the case above both of these pieces of information were kept **private**, and so Eve has to solve a **hard problem** to get access to the shared key!

This kind of philosophy revolutionised Cryptography in the 1970's, and is still a driving theme in today's cryptographic protocols. Classically, security rested entirely on the problem of communicating a shared key secretly. Now, security could be being based on attackers having to solve hard problems to get their hands on the information. In other words, the fact that **Maths is hard** is a good thing!

How might you go about solving the Diffie-Hellman Problem if you don't know x or y? You might try and recover x from $\overline{g}^x$ (or similarly with y). If we were working in $\mathbb{R}$ you know exactly what you would do...take logs!

Definition 4.7. *Let $p \geq 1$ be prime and $\overline{g} \in (\mathbb{Z}/p\mathbb{Z})^\times$ be a primitive root. A **discrete log of $\overline{h}$ with respect to $\overline{g}$** is $x \in \mathbb{Z}$ such that $\overline{h} = \overline{g}^x$. We write $x = \text{dlog}_{\overline{g}}(\overline{h})$.*

Example 4.8. *Given that $\overline{2} \in (\mathbb{Z}/11\mathbb{Z})^\times$ is a primitive root, compute $\text{dlog}_{\overline{2}}(\overline{3})$ and $\text{dlog}_{\overline{2}}(\overline{10})$*

Solution. We compute the powers of $\overline{2}$:

$$\overline{2}^1 = \overline{2}, \quad \overline{2}^2 = \overline{4}, \quad \overline{2}^3 = \overline{8}, \quad \overline{2}^4 = \overline{5}, \quad \overline{2}^5 = \overline{10},$$
$$\overline{2}^6 = \overline{9}, \quad \overline{2}^7 = \overline{7}, \quad \overline{2}^8 = \overline{3}, \quad \overline{2}^9 = \overline{6}, \quad \overline{2}^{10} = \overline{1}$$

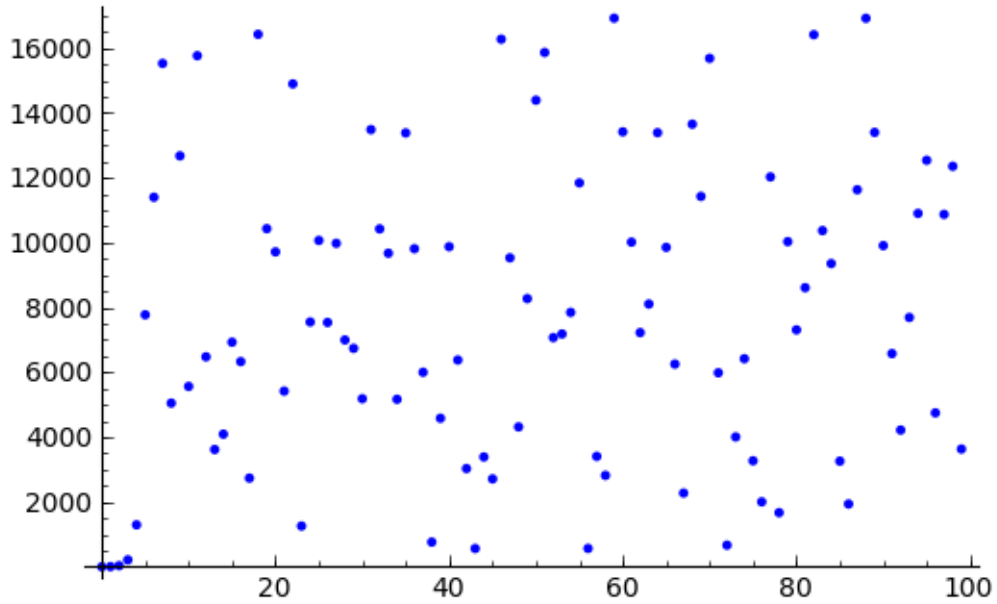
It follows that $\text{dlog}_{\overline{2}}(\overline{3}) = 8$ and $\text{dlog}_{\overline{2}}(\overline{10}) = 5$. ■

Because the powers of $\overline{g}$ repeat every $p-1$ by Fermat's Little Theorem, the quantity $\text{dlog}_{\overline{g}}(\overline{h})$ is only really defined mod $p-1$. But other than that it is unique.

Discrete Log Problem: Given a prime $p \geq 1$, a primitive root $\overline{g} \in (\mathbb{Z}/p\mathbb{Z})^\times$ and $\overline{h} \in (\mathbb{Z}/p\mathbb{Z})^\times$, compute $\text{dlog}_{\overline{g}}(\overline{h})$.

So, if we can solve the Discrete Log Problem efficiently, then we can solve the Diffie-Hellman Problem efficiently. However, this is also a very difficult problem to solve if p is **large**. This is because the powers of $\overline{g} \in (\mathbb{Z}/p\mathbb{Z})^\times$ behave quite **randomly**, as opposed to powers of a real number (where we have a notion of size).

For example, here is a scatter plot of the first 100 powers of the primitive root $\overline{6} \in (\mathbb{Z}/17627\mathbb{Z})^\times$:



The best attacks for the Discrete Log Problem use sophisticated tools from **Algebraic Number Theory**. However, they only run in **sub-exponential** time relative to p and so are not efficient enough to be practical for large prime p (which would be about 600-700 digits long in the real world).

4.2 RSA

Diffie-Hellman Key Exchange provides a way of generating shared keys without needing to communicate them with each other. The idea was to each generate a **private key** and to only communicate **intermediate keys** in public. Gaining access to the shared key required attackers to solve a tricky mathematical problem (the Diffie-Hellman Problem or the Discrete Log Problem).

This was the world's first method of **asymmetric key exchange**. Public knowledge of the intermediate keys did not allow attackers to access either the private keys or the shared keys.

Soon after this, people wondered whether there could be an **asymmetric cipher**, i.e. one where the public **know** how to encrypt messages intended for you, but such that decryption is **only** feasible for the user.

The first such algorithm was announced to the public by Ron **R**ivest, Adi **S**hamir and Leonard **A**dleman in 1977 (although later it was revealed that Clifford Cocks at GCHQ developed the same idea internally in 1973). Their algorithm became known as **RSA**.

Before we see how RSA works, we need to further our understanding of Modular Arithmetic.

Definition 4.9. *Let $N \geq 1$. The group of **units modulo N** is*

$$(\mathbb{Z}/N\mathbb{Z})^\times = \{\overline{a} \in \mathbb{Z}/N\mathbb{Z} : gcd(a, N) = 1\} \subseteq \mathbb{Z}/N\mathbb{Z}.$$

For example, if p is prime then $(\mathbb{Z}/p\mathbb{Z})^\times = \{\overline{1}, \overline{2}, ..., \overline{p-1}\}$ is the group of units mod p from earlier. If $N = 10$ then $(\mathbb{Z}/10\mathbb{Z})^\times = \{\overline{1}, \overline{3}, \overline{7}, \overline{9}\}$.

Definition 4.10. *Let $N \geq 2$. Then the **Euler totient function** is defined by $\phi(N) = |(\mathbb{Z}/N\mathbb{Z})^\times|$.*

We'll see more about this function later. For now we will just need the fact that $\phi(pq) = (p-1)(q-1)$ if $p \neq q \geq 1$ are primes. We will also define $\phi(1) = 1$ where necessary.

It turns out that there is a generalisation of Fermat's Little Theorem for powers in $(\mathbb{Z}/N\mathbb{Z})^\times$.

Theorem 4.11. *(Fermat-Euler Theorem) Let $N \geq 1$ and $a \in \mathbb{Z}$ be such that $gcd(a, N) = 1$. Then $a^{\phi(N)} \equiv 1 \bmod N$.*

Proof. This is very similar to the proof of Fermat's Little Theorem.

Since $\gcd(a, N) = 1$, $\overline{a} \in \mathbb{Z}/N\mathbb{Z}$ is invertible, and so multiplication by $\overline{a}$ is a bijection on $(\mathbb{Z}/N\mathbb{Z})^\times$.

It follows that:

$$\prod_{\overline{x} \in (\mathbb{Z}/N\mathbb{Z})^\times} \overline{ax} = \prod_{\overline{x} \in (\mathbb{Z}/N\mathbb{Z})^\times} \overline{x}.$$

However, this product also equals $\overline{a}^{\phi(N)} \prod_{\overline{x} \in (\mathbb{Z}/N\mathbb{Z})^\times} \overline{x}$, and so we have that

$$\overline{a}^{\phi(N)} \prod_{\overline{x} \in (\mathbb{Z}/N\mathbb{Z})^\times} \overline{x} = \prod_{\overline{x} \in (\mathbb{Z}/N\mathbb{Z})^\times} \overline{x}.$$

But each $\overline{x} \in (\mathbb{Z}/N\mathbb{Z})^\times$ is invertible, so cancelling from both sides gives $\overline{a}^{\phi(N)} = \overline{1}$. □

We'll need to know one more thing before we can state the RSA encryption algorithm.

Lemma 4.12. *Let $N \geq 1$ and $e \in \mathbb{Z}$. The map*

$$f_e : (\mathbb{Z}/N\mathbb{Z})^\times \longrightarrow (\mathbb{Z}/N\mathbb{Z})^\times$$
$$\overline{m} \mapsto \overline{m}^e$$

has an inverse whenever $gcd(e, \phi(N)) = 1$.

Proof. First, note that if $\gcd(e, \phi(N)) = 1$ then there exists $d \in \mathbb{Z}$ such that $ed \equiv 1 \bmod \phi(N)$. Equivalently, $ed = 1 + k\phi(N)$ for some $k \in \mathbb{Z}$.

We claim that the inverse of f_e is f_d. This follows since:

$$f_d(f_e(\overline{m})) = f_d(\overline{m}^e) = \overline{m}^{ed} = \overline{m}^{1+k\phi(N)} = \overline{m} \cdot (\overline{m}^{\phi(N)})^k = \overline{m} \cdot \overline{1}^k = \overline{m},$$

by the Fermat-Euler Theorem. Similarly $f_e(f_d(\overline{m})) = \overline{m}$. □

RSA Encryption/Decryption

1. Alice secretly chooses two distinct primes p and q and computes $N = pq$. She chooses $e \in \mathbb{Z}$ such that $\gcd(e, (p-1)(q-1)) = 1$.
2. Alice publicly reveals her **public key** to be the pair (N, e).
3. Bob **encrypts** the message $\overline{m} \in (\mathbb{Z}/N\mathbb{Z})^\times$ by computing $\overline{c} = f_e(\overline{m}) \in (\mathbb{Z}/N\mathbb{Z})^\times$. He sends this to Alice.
4. To decrypt, Alice uses Euclid's algorithm to compute her **private key**, $d \in \mathbb{Z}$ such that $ed \equiv 1 \bmod (p-1)(q-1)$. She then **decrypts** by computing $\overline{m} = f_d(\overline{c})$.

Note that Alice now has a **public key** (N, e) and a **private key** (p, q, d). The public key allows anyone to communicate with Alice, but the private key is kept secret for decryption purposes.

Proposition 4.13. *The above protocol allows Alice to retrieve the message.*

Proof. Since $\phi(N) = (p-1)(q-1)$, and $\gcd(e, (p-1)(q-1)) = 1$, the map f_e has an inverse. As in the above proof, this is given by f_d with $d \in \mathbb{Z}$ satisfying $ed \equiv 1 \bmod (p-1)(q-1)$.

A simple computation now shows that

$$f_d(\overline{c}) = f_d(f_e(\overline{m})) = \overline{m},$$

so that Alice is able to recover the message. □

Example 4.14. *Alice uses RSA with public key $(N, e) = (55, 3)$. Bob wishes to send the message $\overline{m} = \overline{12} \in (\mathbb{Z}/N\mathbb{Z})^\times$ to Alice.*

What is the ciphertext $\overline{c} \in (\mathbb{Z}/55\mathbb{Z})^\times$ that Bob sends? Demonstrate that Alice is able to decrypt the ciphertext.

Solution. Bob sends the ciphertext:

$$c \equiv m^3 \equiv 12^3 \equiv 1728 \equiv \mathbf{23} \bmod 55.$$

Alice knows that $p = 5$ and $q = 11$ and so computes $(p-1)(q-1) = 40$. She then finds that her private key is $d = \mathbf{27}$, since $3 \cdot 27 \equiv 1 \bmod 40$ (in practice, Euclid's algorithm would be used here).

She recovers the message by computing:

$$m \equiv c^d \equiv 23^{27} \equiv 12^9 \equiv 9 \cdot 8 \cdot (-4)^3 \equiv 17 \cdot (-9) \equiv \mathbf{12} \bmod 55.$$

■

So why is RSA secure? Well once again let's put ourselves in Eve's shoes. The only information available is public key (N, e). If she intercepts a ciphertext $\overline{c}$ then she has to solve the following problem.

RSA Problem: Given $N = pq$ with p, q distinct primes, $e \in \mathbb{Z}$ such that $\gcd(e, \phi(N)) = 1$ and $\overline{c} = \overline{m}^e \in (\mathbb{Z}/N\mathbb{Z})^\times$, retrieve the message $\overline{m} \in (\mathbb{Z}/N\mathbb{Z})^\times$.

Generally, it turns out that this problem is very difficult. The only known way of recovering the message $\overline{m}$ is to calculate the private key d (since f_d is the inverse map of f_e).

However, to calculate d you need to solve the congruence $ed \equiv 1 \bmod (p-1)(q-1)$. Eve could easily do this using Euclid's Algorithm if she knew what the number $(p-1)(q-1)$ was. But there seems to be no way to compute this number without knowing what p and q are (as opposed to just knowing their product, $N = pq$). In other words, Eve needs to solve the following problem.

Semiprime Factorisation Problem: Given a semiprime $N = pq$, recover the primes p and q.

If we can solve the Semiprime Factorisation Problem efficiently, then we can solve the RSA Problem efficiently. However, generally this is also a very difficult problem to solve if p and q are **large**.

The best attacks for the Semiprime Factorisation Problem use sophisticated tools from **Algebraic Number Theory**. However, they only run in **sub-exponential** time relative to N and so are not efficient to be practical for large primes p, q (which would be about $300 - 400$ digits long in the real world).

4.3 El-Gamal

In the last two sections we have got a feel for the idea of public key cryptography. We can now use Mathematics to provide security by forcing attackers to solve hard mathematical problems to recover our messages.

We saw that for the Diffie-Hellman key exchange the security relied on the fact that the Discrete Log Problem is hard to solve. Once this method has been used, both parties use the shared key to encrypt/decrypt under some symmetric scheme. The question now is whether there exists an asymmetric scheme allowing communication, whose security is based on the difficulty of the Discrete Log Problem?

The answer is **yes**, the El-Gamal scheme! This scheme can be used with **any finite group**, and the security relies on the generalised Discrete Log Problem for such groups. In this course, we'll only work with **cyclic groups**. Not much is lost by making this restriction.

First we need to know what discrete logs are in this setting. Let's go back to the case of $(\mathbb{Z}/p\mathbb{Z})^\times$. A discrete log of $\overline{h} \in (\mathbb{Z}/p\mathbb{Z})^\times$ only made sense once we fixed a primitive root $\overline{g} \in (\mathbb{Z}/p\mathbb{Z})^\times$, and then it was defined to be any $b \in \mathbb{Z}$ such that $a = g^b$. For arbitrary cyclic groups things are pretty much identical.

Definition 4.15. *Let G be a finite cyclic group. A **discrete log** of $h \in G$ with respect to generator $g \in G$ is $k \in \mathbb{Z}$ such that $h = g^k$. We write $k = \mathrm{dlog}_g(h)$.*

Remark 4.16. *Be careful, when we write g^k this means apply the group operation k times to g. It doesn't necessarily mean that it's the product of g with itself k times (e.g. the group operation might be addition).*

Recall that the discrete log in $(\mathbb{Z}/p\mathbb{Z})^\times$ was only defined mod $p-1$ (because $\overline{g}^{p-1} = \overline{1}$). Something very similar happens for cyclic groups.

Theorem 4.17. *Let G be a finite cyclic group and $h \in G$. Then the set of discrete logs for $h \in G$ with respect to generator $g \in G$ is a congruence class mod $|G|$.*

Proof. If $k \in \mathbb{Z}$ is a discrete log for h with respect to g then so is $k + |G|m$ for any $m \in \mathbb{Z}$. This follows from the fact that $g^{|G|} = e$:

$$g^{k+|G|m} = g^k g^{|G|m} = g^k (g^{|G|})^m = g^k e^m = h.$$

Now suppose that $k' \in \mathbb{Z}$ is another discrete log for h with respect to g, but that k' is not in the same class as k mod $|G|$. Without loss of generality we can assume that $1 \leq k < k' \leq |G|$. Then $h = g^k = g^{k'}$, implying that $g^{k'-k} = e$. But this contradicts the fact that g is a primitive root, since then g has order dividing $k' - k < |G|$. □

Example 4.18. *Let $G = (\mathbb{Z}/p\mathbb{Z})^\times$. Then the discrete log of $\overline{h} \in G$ with respect to generator $\overline{g} \in G$ is by definition the discrete log that we studied earlier.*

For example, if $p = 11$, $\overline{g} = \overline{2} \in (\mathbb{Z}/11\mathbb{Z})^\times$ and $h = \overline{6} \in (\mathbb{Z}/11\mathbb{Z})^\times$ then $\mathrm{dlog}_g(h) = \mathrm{dlog}_{\overline{2}}(\overline{6}) = 9 + 10\mathbb{Z}$, since $2^9 \equiv 2^{-1} \equiv 6 \bmod 11$.

Example 4.19. *Let $G = \mathbb{Z}/10\mathbb{Z}$ under addition, $g = \overline{7} \in \mathbb{Z}/10\mathbb{Z}$ and $h = \overline{3} \in \mathbb{Z}/10\mathbb{Z}$. Then $\mathrm{dlog}_g(h) = \mathrm{dlog}_{\overline{7}}(\overline{3}) = 9 + 10\mathbb{Z}$, since $7 \cdot 9 = 63 \equiv 3 \bmod 10$.*

Example 4.20. *Let $G = \mu_{10}$, the group of 10th roots of unity under multiplication. The discrete log of $h = e^{\frac{6\pi i}{10}}$ with respect to $g = e^{\frac{14\pi i}{10}}$ is $\mathrm{dlog}_g(h) = 9 + 10\mathbb{Z}$, since $g^9 = e^{\frac{126\pi i}{10}} = e^{\frac{6\pi i}{10}} = h$.*

You might not have realised this but all three of the above calculations were for the **"same"** group (all cyclic of size 10), but the calculations **feel** completely different. This shows that even though two cyclic groups may be isomorphic, discrete log calculations can both look and be completely different.

This all might seem weird (at least to an algebraist)...since we usually only care about groups up to isomorphism. However, in this case it really does matter how the group is **presented** to you and what the group operation is. More on this later...

We are now able to see how discrete logs can be used in public key cryptography.

El-Gamal Encryption

1. Alice chooses a finite cyclic group G of size q and fixes a generator $g \in G$. She chooses a secret value $1 \leq k \leq q$ and computes $h = g^k$. Alice's public key is (G, q, g, h) and her private key is k.
2. Bob encrypts a message $m \in G$ by choosing a secret value $1 \leq s \leq q$ and computing $c_1 = g^s$ and $c_2 = mh^s$ (which uses public information). He sends $c = (c_1, c_2)$ to Alice.
3. Alice decrypts by computing $m = c_2 c_1^{-k}$ (possible since she knows k).

Before seeing an example we first prove that Alice does indeed decrypt to the correct message m.

Lemma 4.21. *The above protocol allows Alice to read the message.*

Proof. This is a simple calculation:

$$c_2 c_1^{-k} = (mh^s)(g^s)^{-k} = (mg^{sk})(g^{-sk}) = m(g^{sk}g^{-sk}) = me = m.$$

□

Example 4.22. *Let's work with the multiplicative group $G = (\mathbb{Z}/31\mathbb{Z})^\times$. Alice chooses generator $g = \overline{3} \in G$ and private key $k = 11 \in \mathbb{Z}$. Then Alice's public key is $(G, q, g, h) = (G, 30, 3, 13)$ since:*

$$h = 3^{11} \equiv (-4)^4 \cdot 3^{-1} \equiv 2 \cdot 4 \cdot 21 \equiv -80 \equiv \mathbf{13} \bmod 31.$$

Bob wishes to send the message $m = \overline{7} \in G$ to Alice. He chooses secret value $s = 4$ and computes:

$$\begin{aligned} c_1 &= 3^4 \equiv \mathbf{19} \bmod 31 \\ c_2 &= 7 \cdot 13^4 \equiv 7 \cdot 14^2 \equiv 70 \equiv \mathbf{8} \bmod 31. \end{aligned}$$

and so sends $(c_1, c_2) = (19, 8)$ to Alice.

Alice decrypts by computing:

$$m = 8 \cdot 19^{-11} \equiv 8 \cdot 19^{19} \equiv 8 \cdot 19 \cdot 20^9 \equiv 8 \cdot 19 \cdot 4^9 \cdot 5^9 \equiv 8 \cdot 19 \cdot 2^3 \equiv 2 \cdot 19 \equiv \mathbf{7} \bmod 31.$$

Remark 4.23. *A few things:*

- *Given an **arbitrary** finite group G we can easily construct cyclic subgroups by picking a random $g \in G$ and taking $H = \langle g \rangle$. In practice, this is usually a good source of cyclic groups for use in El-Gamal.*

- *We viewed messages as elements of $m \in G$, but we didn't explain how we were able to do this. There are good methods of translating conventional messages into group elements, but we won't discuss these in this course.*
- *You might wonder why we include q in the public key. This is mainly since the size of G might not be obvious from the way that G is presented to us.*

We can now discuss the security of the El-Gamal scheme. Note that Eve knows G, q, g, h, c_1 and c_2, and so to get the message m she needs to use this to compute $m = c_2 c_1^{-k}$. However, Eve does not know the value of k and so the only clear way for her to get this is to calculate $\text{dlog}_g(h) = k + q\mathbb{Z}$. But this is an instance of the Discrete Log Problem for G.

The Discrete Log Problem (for cyclic groups): Given a finite cyclic group G, a generator $g \in G$ and $h \in G$, compute $\text{dlog}_g(h)$.

So the security of El-Gamal depends entirely on how hard it is to solve the Discrete Log Problem for the chosen group G. As we saw earlier though, groups that are abstractly the same (i.e. isomorphic) can have Discrete Log Problems of varying difficulty.

Example 4.24. *Let $G = \mathbb{Z}/N\mathbb{Z}$ (under addition) and choose $g \in G$ coprime with N. Then g generates G.*

*Solving the Discrete Log Problem for this setup means being able to solve the congruence $gx \equiv h \bmod N$ (since the group is additive). This congruence is easily solvable by multiplying both sides by $g^{-1} \bmod N$ (computable in polynomial time by **Euclid's algorithm**).*

For example, suppose $N = 13$ and $g = \overline{7}$. Then $\text{dlog}_{\overline{7}}(\overline{9}) = 5 + 13\mathbb{Z}$, since $7x \equiv 9 \bmod 13$ has solution $x \equiv 9 \cdot 7^{-1} \equiv 18 \equiv 5 \bmod 13$. Surely enough $7 + 7 + 7 + 7 + 7 = 7 \cdot 5 = 35 \equiv 9 \bmod 13$.

Given the above example it should be clear that you should never use $G = \mathbb{Z}/N\mathbb{Z}$ with the El-Gamal scheme. The Discrete Log Problem is extremely simple to solve since we have Euclid's algorithm. In stark contrast, we saw that the Discrete Log Problem for $(\mathbb{Z}/p\mathbb{Z})^\times$ is believed to be difficult (even though this group is isomorphic to the additive group $\mathbb{Z}/(p-1)\mathbb{Z}$). Multiplicative groups $(\mathbb{Z}/p\mathbb{Z})^\times$ have proved to be good choices for G in practice.

Another popular choice is $G = E(\mathbb{F}_p)$, the set of points on an **elliptic curve** mod p. The group law here is much less easy to understand, since it's defined geometrically. **Elliptic Curve Cryptography** is used in many real world protocols (e.g. ECDSA, ECDH and Bitcoin).

4.4 Digital Signatures

We have seen now that public key cryptography gives us a great way to communicate securely in an asymmetric way. But it can do much more than this!

One issue with symmetric schemes is that if Eve can find the shared key she can read all messages between Alice and Bob. Also, Eve can then pretend to be Alice or Bob without immediate detection!

Most public key schemes allow the sender of the message to add a **digital signature** providing **authentication** that they were indeed the sender of the message.

The idea is to make the signature hard for Eve to forge. Let's think about how we can do this.

Bad Idea:

Alice thinks for a while and realises that she could probably convince Bob that she is the true sender of a message m by providing something that only she could feasibly know, i.e. her private key.

There are two obvious problems here. Firstly, Bob doesn't know Alice's private key, and so there is no way to **verify** that this really is Alice's secret information.

Secondly, **no one** except Alice should ever be given access to Alice's private key. Otherwise all of her messages can be read!

Good Idea:

Alice realises she needs to show Bob that the sender of the message knows her private key; without telling him or anyone else what it is.

To sign she asks Bob to challenge her to **decrypt** an auxiliary message m_0 using her private key (something that only Alice can do). She then sends her message m (encrypted using Bob's public key) and sign with (m_0, m_1), where m_1 is the decrypted message.

When Bob receives everything he can **verify** the signature by encrypting m_1 using Alice's public key (which everyone knows) and checking that the output does indeed match m_0. He concludes that the sender is probably Alice, since **no one else** can feasibly decrypt a message using only Alice's public key.

Let us now look at two possible digital signature schemes based on the RSA and El-Gamal schemes.

RSA signature scheme

1. Alice chooses an RSA public and private key.
   - **Public key:** (N, e) for some semiprime N and $1 \leq e \leq N$ coprime to $\phi(N)$
   - **Private key:** $d \equiv e^{-1} \mod (p-1)(q-1)$ where p, q are primes such that $N = pq$.
2. Bob chooses an auxiliary message $\overline{m}_0 \in (\mathbb{Z}/N\mathbb{Z})^\times$ and asks Alice to **decrypt** it using her private key. When Alice sends Bob an encrypted message (using his public key) she signs the message by sending the pair $(\overline{m}_0, \overline{m}_1) = (\overline{m}_0, f_d(\overline{m}_0))$ (where d is her private key).
3. Bob can decrypt the encrypted message using his private key and verify Alice's signature by computing $f_e(\overline{m}_1)$ and checking it equals $\overline{m}_0$.

Example 4.25. *Alice chooses primes $p = 5$ and $q = 13$, giving $N = 65$. She then chooses $e = 11$ (which is coprime with $\phi(N) = 4 \cdot 12 = 48$). Her **public key** is then $(N, e) = (65, 11)$ and her **private key** is*

$$d \equiv e^{-1} \equiv 11^{-1} \equiv \mathbf{35} \bmod 48.$$

*Bob chooses secret primes $p' = 7$ and $q' = 13$, giving $N' = 91$. He then chooses $e' = 5$ (which is coprime with $\phi(N') = 6 \cdot 12 = 72$). His **public key** is then $(N', e') = (91, 5)$ and his **private key** is*

$$d' \equiv e'^{-1} \equiv 5^{-1} \equiv \mathbf{29} \bmod 72.$$

Now Alice wishes to send $\overline{m} = \overline{6} \in (\mathbb{Z}/N'\mathbb{Z})^\times$ to Bob using his public key. She computes and sends,

$$c \equiv m^{e'} \equiv 6^5 \equiv 6^3 \cdot 6^2 \equiv 34 \cdot 6 \cdot 6 \equiv 22 \cdot 6 \equiv \mathbf{41} \bmod 91.$$

Bob asks Alice to sign her message by giving her auxiliary message $\overline{m}_0 = \overline{2} \in (\mathbb{Z}/N\mathbb{Z})^\times$. To sign her message Alice uses her private key and computes,

$$m_1 \equiv m_0^d \equiv 2^{35} \equiv (2^7)^5 \equiv (-2)^5 \equiv -32 \equiv \mathbf{33} \bmod 65.$$

Bob receives the encrypted message $\overline{c} = \overline{41} \in (\mathbb{Z}/N'\mathbb{Z})^\times$ and signature $(\overline{m}_0, \overline{m}_1) = (\overline{2}, \overline{33})$. To decrypt the message he uses his private key and computes,

$$\begin{aligned} m = c^{d'} = 41^{29} &\equiv (41^2)^{14} \cdot 41 \equiv (43^2)^7 \cdot 41 \equiv (29^2)^3 \cdot 29 \cdot 41 \\ &\equiv 22^2 \cdot 22 \cdot 29 \cdot 41 \equiv 29 \cdot 22 \cdot 29 \cdot 41 \equiv 29 \cdot 41 \equiv \mathbf{6} \bmod 91. \end{aligned}$$

To verify the signature he uses Alice's public key to compute,

$$m_1^e = 33^{11} \equiv (33^2)^5 \cdot 33 \equiv (49^2)^2 \cdot 49 \cdot 33 \equiv (-4)^2 \cdot 49 \cdot 33 \equiv 16 \cdot 57 \equiv \mathbf{2} \bmod 65.$$

This is the auxiliary message $\overline{m}_0$ and so the signature is verified.

Example 4.26. *Alice chooses RSA public key $(N, e) = (143, 13)$ for primes $p = 11$ and $q = 13$.*

Alice's private key is $d = 37$, since $\phi(N) = 10 \cdot 12 = 120$ and $ed \equiv 13 \cdot 37 = 481 \equiv \mathbf{1} \bmod 120$.

Bob chooses RSA public key $(N', e') = (55, 23)$ for primes $p' = 5$ and $q' = 11$.

Bob's private key is $d' = 7$, since $\phi(N) = 4 \cdot 10 = 40$ and $e'd' \equiv 23 \cdot 7 \equiv 161 \equiv \mathbf{1} \bmod 40$.

Alice wishes to send $\overline{m} = \overline{2} \in (\mathbb{Z}/N'\mathbb{Z})^\times$ to Bob using his public key. She sends $\overline{c} = \overline{8} \in (\mathbb{Z}/N'\mathbb{Z})^\times$ since:

$$c \equiv m^{e'} \equiv 2^{23} \equiv (2^6)^4 \cdot 2^{-1} \equiv 9^4 \cdot 28 \equiv 26^2 \cdot 28 \equiv 13^2 \cdot 2^2 \cdot 28 \equiv 4 \cdot 4 \cdot 28 \equiv 4 \cdot 2 \equiv \mathbf{8} \bmod 55.$$

Bob asks Alice to sign her message by choosing auxiliary message $\overline{m}_0 = \overline{17} \in (\mathbb{Z}/N\mathbb{Z})^\times$. She signs $(\overline{m}_0, \overline{m}_1) = (\overline{17}, \overline{30})$, since:

$$\begin{aligned} m_1 \equiv m_0^d \equiv 17^{37} &\equiv (17^2)^{18} \cdot 17 \equiv 3^{18} \cdot 17 \equiv (3^6)^3 \cdot 17 \equiv (14)^3 \cdot 17 \equiv 7^3 \cdot 2^3 \cdot 17 \\ &\equiv 57 \cdot 8 \cdot 17 \equiv 57 \cdot (-7) \equiv -399 \equiv \mathbf{30} \bmod 143. \end{aligned}$$

Bob receives the encrypted message $\overline{c} = \overline{8} \in (\mathbb{Z}/N'\mathbb{Z})^\times$ and signature $(\overline{m}_0, \overline{m}_1) = (\overline{17}, \overline{30})$. First he uses his private key decrypt the message:

$$m \equiv c^{d'} \equiv 8^7 \equiv (8^2)^3 \cdot 8 \equiv 9^3 \cdot 8 \equiv 26 \cdot 9 \cdot 8 \equiv 26 \cdot 17 \equiv 442 \equiv \mathbf{2} \bmod 55.$$

He verifies Alice's signature using her public key:

$$\begin{aligned} m_1^e &\equiv 30^{13} \equiv 2^{13} \cdot 3^{13} \cdot 10^{13} \equiv 3^{13} \cdot 10 \equiv (3^6)^3 \cdot 3 \cdot 10 \equiv 14^2 \cdot 3 \cdot 10 \\ &\equiv 53 \cdot 3 \cdot 10 \equiv 16 \cdot 10 \equiv 160 \equiv \mathbf{17} \bmod 143. \end{aligned}$$

In practice, Alice can use the actual message m in order to sign. She clearly cannot take $\overline{m}_0 = \overline{m}$ (otherwise everyone would see the message!) but can instead apply a **hash function** H to $\overline{m}$ and take $\overline{m}_0 = H(\overline{m})$. Hash functions are functions that produce a **short**, fixed length string from **variable length** messages in such a way that is hard to invert, or **forge**.

While the RSA signature scheme is secure (if the parameters are chosen well), it might worry some people that they are always using their private key to sign the message. Over time Eve will gain information about this. For this reason Alice might use a second set of keys specifically designed for **signing**.

We can also make signature schemes that complement the El-Gamal scheme, although these are a little more complicated than RSA signatures. The signature scheme we will see only works for the cyclic groups $G = (\mathbb{Z}/p\mathbb{Z})^\times$ with p prime (which have size $q = p - 1$).

El-Gamal signature scheme

1. Alice chooses an El-Gamal public and private key.
   - **Public key:** (G, q, g, h) where $G = (\mathbb{Z}/p\mathbb{Z})^\times$ for some prime p and $q = |G| = p - 1$, $g \in G$ a primitive root and $h = g^k$.
   - **Private key:** $1 \leq k \leq p - 1$.
2. Alice chooses an auxiliary message $m_0 \in G$ and secret value $1 \leq s \leq q$ that is coprime to q. She computes
   $$m_1 = g^s \quad \text{and} \quad m_2 \equiv (m_0 - km_1)s^{-1} \bmod q.$$
   When she sends an encrypted message to Bob (using his public key) she signs by also sending (m_0, m_1, m_2).
3. Bob decrypts the message using his private key and verifies Alice's signature by computing $h^{m_1} m_1^{m_2}$ and checking that it equals g^{m_0}.

Proposition 4.27. *The verification step is theoretically correct.*

Proof. We see this is true since,

$$h^{m_1} m_1^{m_2} = (g^k)^{m_1} (g^s)^{m_2} = g^{km_1 + sm_2} = g^{km_1 + s(m_0 - km_1)s^{-1}} = g^{m_0}.$$

□

Example 4.28. *Alice chooses prime $p = 17$, primitive root $g = 6 \in G = (\mathbb{Z}/p\mathbb{Z})^\times$ and **private key** $k = 8$. She computes,*

$$h \equiv g^k \equiv 6^8 \equiv (6^2)^4 \equiv 2^4 \equiv \mathbf{16} \bmod 17.$$

Alice's ***public key*** *is then* $(G, q, g, h) = (G, 16, 6, 16)$.

Bob chooses prime $p' = 23$*, primitive root* $g' = 11 \in G' = (\mathbb{Z}/p'\mathbb{Z})^\times$ *and* ***private key*** $k' = 4$*. He computes,*

$$h' \equiv g'^{k'} \equiv 11^4 \equiv (121)^2 \equiv 6^2 \equiv \mathbf{13} \bmod 23.$$

Bob's ***public key*** *is then* $(G', q', g', h') = (G', 22, 11, 13)$.

Alice wishes to send Bob the message $m = 16 \in G'$*. To encrypt she uses Bob's public key and a secret value* $s_1 = 3$ *to compute,*

$$\begin{aligned} c_1 &\equiv g'^{s_1} \equiv 11^3 \equiv 6 \cdot 11 \equiv \mathbf{20} \bmod 23 \\ c_2 &\equiv mh'^{s_1} \equiv 16 \cdot 13^3 \equiv 16 \cdot 8 \cdot 13 \equiv 13 \cdot 13 \equiv \mathbf{8} \bmod 23. \end{aligned}$$

To sign her message Alice choose auxiliary message $m_0 = \mathbf{5} \in G$ *and secret value* $s_2 = 7$*. She computes,*

$$\begin{aligned} m_1 &\equiv g^{s_2} \equiv 6^7 \equiv (6^2)^3 \cdot 6 \equiv 2^3 \cdot 6 \equiv 8 \cdot 6 \equiv \mathbf{14} \bmod 17 \\ m_2 &\equiv (m_0 - km_1)s_2^{-1} \equiv (5 - 8 \cdot 14) \cdot 7^{-1} \equiv (5 - 0) \cdot 7 \equiv \mathbf{3} \bmod 16. \end{aligned}$$

Bob receives the encrypted message $c = (c_1, c_2) = (20, 8)$ *and signature* $(m_0, m_1, m_2) = (5, 14, 3)$*, To decrypt the message he uses his private key to compute:*

$$\begin{aligned} m \equiv c_2 c_1^{-k'} \equiv 8 \cdot 20^{-4} \equiv 8 \cdot 20^{18} &\equiv 8 \cdot (20^2)^9 \equiv 8 \cdot (9^2)^4 \cdot 9 \equiv 8 \cdot (12^2)^2 \cdot 9 \\ &\equiv 8 \cdot 6^2 \cdot 9 \equiv 8 \cdot 13 \cdot 9 \equiv 12 \cdot 9 \equiv \mathbf{16} \bmod 23. \end{aligned}$$

To verify Alice's signature Bob uses her public key to compute,

$$\begin{aligned} h^{m_1} m_1^{m_2} &\equiv 16^{14} \cdot 14^3 \equiv (-1)^{14} \cdot (-3)^3 \equiv 1 \cdot (-27) \equiv \mathbf{7} \mod 17 \\ g^{m_0} &\equiv 6^5 \equiv (6^2)^2 \cdot 6 \equiv 2^2 \cdot 6 \equiv \mathbf{7} \mod 17. \end{aligned}$$

The two values are equal and so the signature is verified.

Example 4.29. *Alice chooses a El-Gamal public key of the form* $(G, q, g, h) = (G, 22, 5, h)$ *and chooses private key* $k = 11$*. She calculates that* $h = 22$*, since:*

$$h \equiv g^k \equiv 5^{11} \equiv (5^2)^5 \cdot 5 \equiv 2^5 \cdot 5 \equiv 9 \cdot 5 \equiv \mathbf{22} \bmod 23.$$

Bob chooses El-Gamal public key of the form $(G', q', g', h') = (G', 16, 3, h')$ *and chooses private key* $k' = 7$*. He calculates that* $h' = 11$*, since:*

$$h' \equiv g'^{k'} \equiv 3^7 \equiv (3^3)^2 \cdot 3 \equiv 10^2 \cdot 3 \equiv (-2) \cdot 3 \equiv -6 \equiv \mathbf{11} \bmod 17.$$

Alice wishes to send Bob message $m = 13 \in G'$ *which she encrypts using his public key and secret value* $s_1 = 3$*. She sends* $(c_1, c_2) = (10, 14)$ *to Bob, since:*

$$\begin{aligned} c_1 &\equiv g'^{s_1} \equiv 3^3 \equiv \mathbf{10} \bmod 17 \\ c_2 &\equiv mh'^{s_1} \equiv 13 \cdot 11^3 \equiv (-4) \cdot (-6)^3 \equiv 24 \cdot (-6)^2 \equiv 7 \cdot 2 \equiv \mathbf{14} \bmod 17. \end{aligned}$$

Alice also signs the message by choosing auxiliary message $m_0 = \mathbf{19} \in G$ and secret value $s_2 = 9$. She signs with $(m_0, m_1, m_2) = (19, 11, 18)$, since:

$$m_1 \equiv g^{s_2} \equiv 5^9 \equiv (5^2)^4 \cdot 5 \equiv 2^4 \cdot 5 \equiv 16 \cdot 5 \equiv 80 \equiv \mathbf{11} \bmod 23$$
$$m_2 \equiv (m_0 - km_1)s_2^{-1} \equiv (19 - 11 \cdot 11) \cdot 9^{-1} \equiv (-102) \cdot 5 \equiv (-14) \cdot 5 \equiv -70 \equiv \mathbf{18} \bmod 22.$$

Bob receives the encrypted message $(c_1, c_2) = (10, 14)$ and signature $(m_0, m_1, m_2) = (19, 11, 18)$. To decrypt the message he uses his private key:

$$m \equiv c_2 c_1^{-k'} \equiv 14 \cdot 10^{-7} \equiv (-3) \cdot 12^7 \equiv (-3) \cdot ((-5)^2)^3 \cdot (-5) \equiv 15 \cdot 8^3 \equiv (-2) \cdot 8 \cdot 8^2 \equiv 1 \cdot 64 \equiv \mathbf{13} \mod 17.$$

To verify Alice's signature Bob uses her public key to compute,

$$\begin{aligned} h^{m_1} m_1^{m_2} &\equiv 22^{11} \cdot 11^{18} \equiv (-1)^{11} \cdot (11^2)^9 \equiv (-1) \cdot 6^9 \equiv (-1) \cdot (6^2)^4 \cdot 6 \equiv (-1) \cdot 13^4 \cdot 6 \\ &\equiv (-1) \cdot 8^2 \cdot 6 \equiv (-1) \cdot (-5) \cdot 6 \equiv \mathbf{7} \bmod 23 \\ g^{m_0} &\equiv 5^1 9 \equiv (5^2)^9 \cdot 5 \equiv 2^9 \cdot 5 \equiv 2^5 \cdot 2^4 \cdot 5 \equiv 9 \cdot (-7) \cdot 5 \equiv 45 \cdot (-7) \equiv (-1) \cdot (-7) \equiv \mathbf{7} \bmod 23. \end{aligned}$$

The two values are equal and so the signature is verified.

We will now discuss why this signature scheme is secure. Suppose Eve wants to forge Alice's signature. She knows Alice's public key (G, q, g, h) where $h = g^k$ for some secret k. In order to forge the signature she needs to be able to find $a, d, c \in \mathbb{Z}$ such that $h^b b^c = g^a$. Then she can **fraudulently** sign a message to Bob by sending the triple $(m_0, m_1, m_2) = (a, b, c)$.

We can see that this is hard for Eve to do. If we take logs with respect to g we get

$$b \cdot \mathrm{dlog}_g(h) + c \cdot \mathrm{dlog}_g(b) \equiv a \mod (p-1).$$

The only known way for Eve to solve this is to pick values for a and b, compute $\mathbf{dlog}_g(h)$ and $\mathbf{dlog}_g(b)$ and then solve for c. Even if Eve picks b well she still needs to find $\mathbf{dlog}_g(h)$, which is a **generic** instance of the Discrete Log Problem (as long as Alice chooses her key well). Thus **only** Alice should feasibly be able to sign the messages that she sends.

5 Factorisation methods

We've just seen that the idea of a public key scheme is to base your security on the toughness of a Mathematical problem. For the RSA scheme, this problem was the Semiprime Factorisation Problem. This is believed to be a hard problem in that no efficient algorithm has ever been found to solve it using a classical computer. However, Pure Mathematicians are never happy with "belief", we want to be sure that it's a hard problem!

There are other reasons why we might want to understand in detail how hard a given problem is:

- If we're basing our crypt on a "hard problem" that secretly turns out to be easy...the scheme is rendered useless!
- Knowing how easy/hard a given instance of a problem is might inform us on how we should choose our keys in reality. Capabilities are changing all the time and we have to keep up.
- Studying one hard problem might shed light on another hard problem. Maybe we can solve one by only having to solving the other? Maybe we can start to understand the "ranking" of problems, based on their difficulty?
- Studying hard problems might lead to new Mathematics that is applicable elsewhere.

In this section we will see a variety of simple attacks on the Semiprime Factorisation Problem. Each performs relatively well for small semiprimes, but none perform well enough to tackle real world key sizes.

5.1 Trial division

Trial division is the world's oldest factoring algorithm. If we're given $N = pq$ with $p \neq q$ prime and asked to factorise then we can do as follows:

Trial division: For each $2 \leq i \leq N-1$, check whether $i \mid N$. If so then $i \in \{p, q\}$ is a non-trivial prime factor.

This "try all of the possibilities" approach is guaranteed to work, but is extremely time consuming. We can speed things up a little by only checking divisibility by the primes $p \leq \sqrt{N}$ by the following fact.

Lemma 5.1. *If an integer $M \geq 2$ is composite then $p \mid M$ for some prime $p \leq \sqrt{M}$.*

Proof. Since $M \geq 2$ it has a prime factorisation $M = \prod_{p_i \mid M} p_i^{e_i}$. If every $p_i > \sqrt{M}$ then the RHS is greater than $\sqrt{M}^2 = M$, giving a contradiction. □

Example 5.2. *If $N = 143$ then $\sqrt{143} \approx 11.96$. We see that the first few primes are such that $2, 3, 5, 7 \nmid 143$. However, $11 \mid 143$, giving factorisation $N = 143 = 11 \cdot 13$.*

Even with the speed up this method is very inefficient. The Prime Number Theorem tells us that there are $\pi(\sqrt{N}) \sim \frac{\sqrt{N}}{\log(\sqrt{N})}$ primes $p \leq \sqrt{N}$. This is huge for large N.

5.2 Fermat's method

Fermat's method relies on the age old identity:

$$a^2 - b^2 = (a-b)(a+b).$$

Idea: If we can write $N = a^2 - b^2$ for some $a, b \in \mathbb{N}$ then it's almost guaranteed that $a - b = p$ and $a + b = q$.

Lemma 5.3. *Suppose $N = pq \geq 2$ can be written as $N = a^2 - b^2$ with $a, b \in \mathbb{N}$ and $(a, b) \neq \left(\frac{N+1}{2}, \frac{N-1}{2}\right)$. Then $a \pm b \in \{p, q\}$.*

Proof. We use the difference of two squares to factor N as:

$$N = a^2 - b^2 = (a-b)(a+b).$$

Since $N \geq 2$ we have that $a > b \geq 1$ and so $a \pm b$ are positive divisors of N whose product is N. Since $N = pq$ for two distinct primes p, q, it follows that $\{a-b, a+b\} = \{1, N\}$ or $\{p, q\}$.

The first case can only happen if $a - b = 1$ and $a + b = N$ (since $a - b < a + b$). The solution to this pair of equations is $(a, b) = \left(\frac{N+1}{2}, \frac{N-1}{2}\right)$, and so any other pair (a, b) must correspond to the non-trivial factors p, q. □

How will we find such an a, b? We could instead consider the values $N + b^2$ and wait until we see a square. This gives us an algorithm:

Fermat's method:

For each $1 \leq b \leq \frac{N-3}{2}$ check whether $N + b^2 = a^2$ for some $a \in \mathbb{N}$. If so then $a \pm b \in \{p, q\}$, so we learn the two prime factors.

Example 5.4. *If $N = 143$ then we compute that $N + 1^2 = N + 1 = 144 = 12^2$. It follows that:*

$$N = 143 = 12^2 - 1^2 = (12-1)(12+1) = 11 \cdot 13,$$

revealing the factors 11 and 13.

This method works very well when the two primes are **close together** (regardless of their size). The reason is that the solution to the equations $a - b = p$ and $a + b = q$ is $(a, b) = \left(\frac{p+q}{2}, \frac{p-q}{2}\right)$, and p, q being close is equivalent to b being small (which we would then find early on in our search). For this reason, the primes used to generate an RSA key must be both **large** and **not close**.

The problem with Fermat's method is that there is only **one** value of b that leads to success. This is no good...since it might take a long time to find it. A more general algorithm is as follows:

Fermat's method mod N:

For each $1 \leq a, b \leq N - 1$ check whether $a^2 \equiv b^2 \bmod N$ and $a \not\equiv \pm b \bmod N$. If so then $\gcd(a \pm b, N) \in \{p, q\}$.

This works because of the following fact.

Lemma 5.5. *Suppose $N = pq \geq 2$ and $a^2 \equiv b^2 \bmod N$ with $a \not\equiv \pm b \bmod N$. Then gcd$(a \pm b, N) \in \{p, q\}$.*

Proof. The congruence implies that $(a - b)(a + b) \equiv 0 \bmod N$. Since $a \not\equiv \pm b \bmod N$, neither bracket is $0 \bmod N$. But $N = pq$ implies that the only other solutions are $a - b \equiv 0 \bmod p$ and $a + b \equiv 0 \bmod q$ or visa versa with p, q swapped (in each case, neither is congruent to $0 \bmod N$). It follows that $\gcd(a \pm b, N) \in \{p, q\}$. □

Example 5.6. *If $N = 143$ then we compute that $12^2 \equiv 1^2 \bmod 143$ and $12 \not\equiv 1 \bmod 143$. It follows that $\gcd(12 \pm 1, 143)$ give the non-trivial prime factors of $N = 143$. Indeed $\gcd(11, 143) = 11$ and $\gcd(13, 143) = 13$.*

This method has the advantage that there can be **many** (a, b) pairs that satisfy the congruence. However, we're no closer to finding them.

5.3 The Pollard Rho method for factoring

Pollard's idea for finding a solution to the congruence $a^2 \equiv b^2 \bmod N$ is to **iterate** the map $f(x) \equiv x^2 + 1 \bmod N$ for some starting value $x_0 \in \mathbb{Z}$:

$$x_0, \qquad x_1 = f(x_0) \equiv x_0^2 + 1 \bmod N, \qquad x_2 = f(x_1) \equiv x_1^2 + 1 \bmod N, \qquad ...$$

Since $f(x)$ seemingly behaves randomly mod p, eventually you'll stumble on a collision

$$f(x_i) \equiv f(x_j) \bmod p$$

for some $i < j$, giving $x_i^2 \equiv x_j^2 \bmod p$. Then as before $\gcd(x_i \pm x_j, N)$ is likely to be p.

Baby Pollard Rho:

- Choose a starting value $x_0 \in \mathbb{Z}$.
- Iterate the function $f(x) \equiv x^2 + 1 \bmod N$ to get values $x_0, x_1, x_2, ...$
- Check $\gcd(x_i \pm x_j, N)$ as you go for non-trivial prime factors of N.

Example 5.7. *If $N = 143$ and starting value $x_0 = 2$ we get:*

$$x_1 = f(x_0) \equiv 2^2 + 1 \equiv 5 \bmod 143.$$

Since $\gcd(x_1 \pm x_0, 143) = gcd(7, 143) = gcd(3, 143) = 1$ we must continue.

Now compute:

$$x_2 = f(x_1) \equiv 5^2 + 1 \equiv 26 \bmod 143.$$

Once again, we must continue since $\gcd(x_2 \pm x_0, 143) = \gcd(x_2 \pm x_1, 143) = 1$.

Now compute:

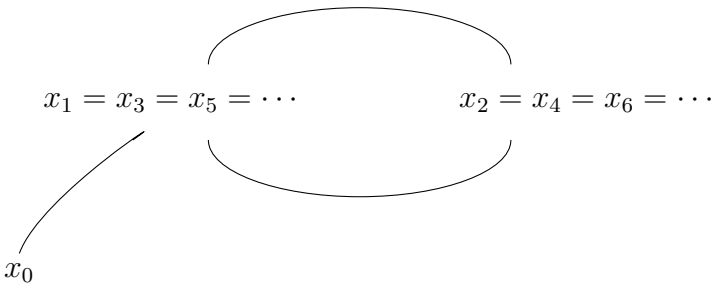
$$x_3 = f(x_2) \equiv 26^2 + 1 \equiv 105 \bmod 143.$$

We immediately check that $\gcd(x_3 + x_1, 143) = \gcd(110, 143) = 11$, revealing a non-trivial prime factor of N.

The reason for the "rho" in the name of the algorithm is that once a collision $f(x_i) \equiv f(x_j) \bmod p$ is found, the values start to cycle mod p, making a rho shape. For example, in the above example we found that $p = 11$ and we see that:

$$x_1^2 + 1 \equiv x_3^2 + 1 \equiv x_5^2 + 1 \equiv ... \equiv 4 \bmod 11$$

$$x_2^2 + 1 \equiv x_4^2 + 1 \equiv x_6^2 + 1 \equiv ... \equiv 6 \bmod 11.$$



The Pollard Rho algorithm can be improved by instead doing the following:

Pollard Rho for factoring:

- Choose starting pair (x_0, y_0) with $x_0 = y_0 \in \mathbb{Z}$.
- Iterate the functions $(x, y) \mapsto (f(x), f(f(y)))$ to get pairs $(x_0, y_0), (x_1, y_1), (x_2, y_2), ...$
- Check $\gcd(x_i \pm y_i, N)$ as you go for non-trivial prime factors of N.

Example 5.8. *If $N = 143$ and starting pair $x_0 = y_0 = 2$ then we get:*

$$x_1 \equiv x_0^2 + 1 \equiv 2^2 + 1 \equiv 5 \bmod 143$$
$$y_1 \equiv (y_0^2 + 1)^2 + 1 \equiv (2^2 + 1)^2 + 1 \equiv 26 \bmod 143.$$

Since $\gcd(x_1 \pm y_1, N) = \gcd(31, 143) = \gcd(-21, 143) = 1$ *we continue.*

Next we get:

$$x_2 \equiv x_1^2 + 1 \equiv 5^2 + 1 \equiv 26 \bmod 143$$
$$y_2 \equiv (y_1^2 + 1)^2 + 1 \equiv (26^2 + 1)^2 + 1 \equiv 15 \bmod 143.$$

This time we find that $\gcd(x_2 - y_2, N) = \gcd(11, 143) = 11$*, giving a non-trivial prime factor of N.*

This cuts down on the number of gcd calculations necessary and actually speeds up the algorithm (although this is not obvious!).

5.4 Dixon's method

Recall that we wish to find a non-trivial solution to $a^2 \equiv b^2 \bmod N$, since then $\gcd(a \pm b, N) \in \{p, q\}$. We've seen two ways to do this, but one requires a long brute force search and the other works quicker...but only on average.

Dixon's method uses **Linear Algebra** to solve the problem, providing ideas which form the basis of some of the best factorisation algorithms that we know (e.g. the Quadratic Sieve and more generally the Number Field Sieve).

The method begins with the simple fact that the product of squares is another square. Let's see a simple example of how this fact can be useful in providing solutions to the above congruence

Example 5.9. *Suppose that we square three random integers $a_1, a_2, a_3 \in \mathbb{Z}$, reduce mod N and factorise the results. Suppose also that we get very lucky and find that:*

$$a_1^2 \equiv b_1 \equiv p_1 p_2 \bmod N$$
$$a_2^2 \equiv b_2 \equiv p_1 p_3 \bmod N$$
$$a_3^2 \equiv b_3 \equiv p_2 p_3 \bmod N,$$

for three distinct primes p_1, p_2, p_3 (not related to N in any way).

None of these congruences solves our problem directly...but together they do!

Multiplying the three congruences gives:

$$(a_0 a_1 a_2)^2 \equiv b_1 b_2 b_3 \equiv (p_1 p_2)(p_1 p_3)(p_2 p_3) \equiv (p_1 p_2 p_3)^2 \bmod N.$$

We have found a (hopefully non-trivial) solution $(a, b) = (a_0 a_1 a_2, p_1 p_2 p_3)$ to the congruence $a^2 \equiv b^2 \bmod N$.

Dixon's idea was to turn the above into a general algorithm. We could square a bunch of numbers $a_1, a_2, ..., a_k$, reduce mod N and factor, getting a bunch of congruences of the form:

$$a_m^2 \equiv \prod_i p_i^{e_{i,m}} \bmod N.$$

If we do enough work then we might be able to find some subset of congruences whose product gives a solution to $a^2 \equiv b^2 \bmod N$. In other words, we might be able to find a product:

$$\left(\prod_{m \in M} a_m \right)^2 \equiv \prod_{m \in M} \prod_i p_i^{e_{i,m}} \equiv \prod_i p_i^{\sum_{m \in M} e_{i,m}} \bmod N,$$

such that each of the powers $\sum_{m \in M} e_{m,i}$ is **even**.

But how will be find such a subset of congruences that can be combined in this way? It was obvious how to do this in the above example, but in general this is not so obvious! It turns out that we can solve this issue by using **Linear Algebra**.

Suppose that we fix an ordering of the primes, $p_1, p_2, ..., p_n, ...$ (e.g. we'll use the natural ordering by size, $2, 3, 5, 7, ...$). Then we can represent the prime factorisation of a rational number as a vector.

We define the following group under addition (see the exercises for the proof of this fact):

$$\bigoplus_{n\geq 1}\mathbb{Z} = \{(x_1, x_2, x_3, ...) \,|\, x_i \in \mathbb{Z} \text{ and } x_i = 0 \text{ for all but finitely many } i\}.$$

Lemma 5.10. *The following map is a group homomorphism:*

$$\psi : \mathbb{Q}^\times \longrightarrow \bigoplus_{n\geq 1}\mathbb{Z}$$
$$n = \prod_i p_i^{e_i} \longmapsto (e_1, e_2, e_3, ...).$$

The image of the subgroup $(\mathbb{Q}^\times)^2$ of squares is the subgroup $\bigoplus_{n\geq 1} 2\mathbb{Z}$.

Proof. The function ψ is well defined since a rational number can only have finitely many primes dividing the numerator/denominator (so that $e_n = 0$ for all but finitely many n, as required).

The first claim follows from the simple fact that if $n = \prod_i p_i^{e_i} \in \mathbb{Q}^\times$ and $m = \prod_i p_i^{f_i} \in \mathbb{Q}^\times$ then:

$$\psi(nm) = \phi\left(\prod_i p_i^{e_i+f_i}\right) = (e_1+f_1, e_2+f_2, e_3+f_3, ...) = (e_1, e_2, e_3, ...)+(f_1, f_2, f_3, ...) = \psi(n)+\psi(m).$$

The second claim follows from the fact that $n^2 = \prod_i p_i^{2e_i} \in \mathbb{Q}^\times$ is sent to $\psi(n^2) = \psi\left(\prod_{i\geq 1} p_i^{2e_i}\right) = (2e_1, 2e_2, 2e_3, ...) \in \bigoplus_{n\geq 1} 2\mathbb{Z}$. Similarly, any element of this direct sum clearly corresponds to a factorisation of some non-zero rational number. □

We now know how to proceed, since we can instead work with the vectors:

$$\mathbf{e}_m = \psi\left(\prod_i p_i^{e_{i,m}}\right) = (e_{1,m}, e_{2,m}, e_{3,m}, ...) \in \bigoplus_{n\geq 1}\mathbb{Z}.$$

Then by the lemma we have:

$$\prod_{m\in M}\prod_i p_i^{e_{i,m}} \in (\mathbb{Q}^\times)^2 \iff \psi\left(\prod_{m\in M}\prod_i p_i^{e_{i,m}}\right) \in \bigoplus_{n\geq 1} 2\mathbb{Z} \iff \sum_{m\in M}\mathbf{e}_m \in \bigoplus_{n\geq 1} 2\mathbb{Z}.$$

Finally, we note that:

$$\sum_{m\in M}\mathbf{e}_m \in \bigoplus_{n\geq 1} 2\mathbb{Z} \iff \sum_{m\in M}\overline{\mathbf{e}}_m = \overline{\mathbf{0}} \in \bigoplus_{n\geq 1}\mathbb{Z}/2\mathbb{Z},$$

where

$$\overline{\mathbf{e}}_m = (\overline{e}_{1,m}, \overline{e}_{2,m}, \overline{e}_{3,m}, ...) \in \bigoplus_{n\geq 1}\mathbb{Z}/2\mathbb{Z},$$

is the reduction mod 2.

The condition $\sum_{m\in M}\overline{\mathbf{e}}_m = \overline{\mathbf{0}}$ should look familiar. If we write our vectors $\overline{e}_m$ as columns in a matrix E with entries mod 2, then this condition corresponds to finding a **null vector**, i.e. a vector $\mathbf{v}$ such that $E\mathbf{v} = \overline{\mathbf{0}}$. This is a problem we know how to solve using **Linear Algebra**!

However, there's one subtle issue that we are forgetting...there are **infinitely many primes**, so that the matrix E is infinite!

Idea: Fix a **finite** set of primes and **only** allow congruences that feature these primes. Then the matrix E will be finite.

Definition 5.11. *Let $B \geq 2$. An integer $n \geq 1$ is B-**smooth** if every prime $p \,|\, n$ satisfies $p \leq B$.*

Example 5.12. *The integer $n = 1024$ is B-smooth for any $B \geq 2$ since $n = 2^{10}$ only has $p = 2$ as a prime factor. However, $n = 1025$ is not 2-smooth since $5|1025$ and $5 > 2$ (this integer is B-smooth if and only if $B \geq 41$).*

Since a B-smooth number n can only be divisible by primes $p \leq B$, we have that $\psi(n)$ is a finite vector of fixed length $\pi(B) = \#\{p \text{ prime} \,|\, p \leq B\}$.

We're now ready to see Dixon's method in generality!

Dixon's method:

1. Choose a smoothness bound $B \geq 2$ and list the primes $p_1, p_2, ..., p_{\pi(B)} \leq B$.
2. Search for integers $a_1, a_2, a_3, ..., a_k > \sqrt{N}$ such that:
$$a_m^2 \equiv b_m \equiv \prod_{1 \leq i \leq \pi(B)} p_i^{e_{i,m}} \bmod N,$$
i.e. with b_m being B-smooth.
3. Form the matrix E whose columns are the first $\pi(B)$ entries of the vectors $\overline{\mathbf{e}}_m$, i.e. the mod 2 matrix:
$$\begin{pmatrix} \overline{e_{1,1}} & \overline{e_{1,2}} & \overline{e_{1,3}} & \cdots & \overline{e_{1,k}} \\ \overline{e_{2,1}} & \overline{e_{2,2}} & \overline{e_{2,3}} & \cdots & \overline{e_{2,k}} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \overline{e_{\pi(B),1}} & \overline{e_{\pi(B),2}} & \overline{e_{\pi(B),3}} & \cdots & \overline{e_{\pi(B),k}} \end{pmatrix}$$
4. Compute $\text{NullSpace}(E) = \{\mathbf{v} \in \mathbb{F}_2^k \,|\, E\mathbf{v} = \overline{\mathbf{0}}\}$ using your favourite technique, e.g. Gaussian Elimination (remembering to work mod 2).
5. Pick a non-zero element $\mathbf{v} \in \text{NullSpace}(E)$ (if this is not possible then continue the search for more integers a_i until this is possible).
6. Compute the product:
$$\prod_{m, v_m = \overline{1}} a_m^2 \equiv \prod_{m, v_m = \overline{1}} b_m \bmod N.$$
By construction the RHS is an integer square and so we find an integer solution
$$(a, b) = \left(\prod_{m, v_m = \overline{1}} a_m \quad , \quad \sqrt{\prod_{m, v_m = \overline{1}} b_m} \right),$$
to the congruence $a^2 \equiv b^2 \bmod N$.
7. Check whether $\gcd(a \pm b, N) \in \{p, q\}$. If not then pick another non-zero null vector $\mathbf{v}$ to try, or start again with another selection of integers a_i.

Example 5.13. *Let's factor $N = 143$ again. Choose smoothness bound $B = 7$. Then:*

$$17^2 \equiv 289 \equiv 3 \bmod 143$$
$$20^2 \equiv 400 \equiv 114 \equiv 2 \cdot 3 \cdot 19 \bmod 143$$
$$27^2 \equiv 729 \equiv 14 \equiv 2 \cdot 7 \bmod 143$$
$$69^2 \equiv 4761 \equiv 42 \equiv 2 \cdot 3 \cdot 7 \bmod 143$$

The second congruence gives a RHS that is not 7-smooth and so we discard it. We now have that $(a_1, a_2, a_3) = (17, 27, 69)$ and $(b_1, b_2, b_3) = (3, 14, 42)$.

The matrix E is:

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

so that $\text{NullSpace}(E) = \{(\overline{a}, \overline{a}, \overline{a}) \mid \overline{a} \in \mathbb{Z}/2\mathbb{Z}\} = \text{Span}_{\mathbb{Z}/2\mathbb{Z}}((\overline{1}, \overline{1}, \overline{1}))$.

It follows that:

$$(a_1 a_2 a_3)^2 \equiv b_1 b_2 b_3 \equiv 3 \cdot 14 \cdot 42 \equiv (2 \cdot 3 \cdot 7)^2 \equiv 42^2 \bmod 143.$$

We then find that $\gcd(a_1 a_2 a_3 - 42, 143) = \gcd(31629, 143) = 13$*, revealing one of the two prime factors of N.*

It might seem like we did a lot of work to factor a number that was small enough to be done by simpler techniques...however, this method really does scale quite well to numbers with much larger prime factors. Its more advanced sibling, the **Number Field Sieve**, uses very similar techniques and is currently the best factorisation algorithm known for large numbers.

We won't analyse the performance of this method in this course, but note that there is a balancing act to be performed:

- As the smoothness bound B **decreases** then the chances of b_m being B-smooth **decreases**, so it takes **longer** to get enough congruences to work with, i.e. to find a matrix E that has a non-zero null vector.
- As the smoothness bound B **increases** we receive **more** potentially useful congruences, but the linear algebra step takes **longer** (since the matrices are getting **larger**).

Choosing an ideal smoothness bound is an art, but enough study has been done on this problem and we know the optimal choice of B. Unfortunately (or fortunately, depending on how you look at it) the run time of this algorithm is still way too high to break real world RSA keys.

6 Discrete Log methods

As mentioned previously, the security of the RSA scheme relies on the toughness of the Semiprime Factorisation Problem. We saw some of the basic attacks on this problem and while some work much better than others, none were competitive with real world public keys.

We now turn to Diffie-Hellman Key Exchange and the El-Gamal scheme. The security of these depends on the Discrete Log Problem for a cyclic group G, the problem of determining $k \in \mathbb{Z}$ given knowledge of a generator $g \in G$ and the element $h = g^k$ (i.e. finding a representative of the coset $\text{dlog}_g(h) = k + |G|\mathbb{Z}$). What are the possible attacks on this problem? We'll take a look at some of the basic ones in this section.

6.1 Brute force

For factoring we always had the option of simply trying all possibilities. That's obviously still an option here too!

Brute force: For each $1 \leq k \leq |G|$ check whether $g^k = h$. If so then $\text{dlog}_g(h) = k + |G|\mathbb{Z}$.

Example 6.1. *Let* $G = (\mathbb{Z}/37\mathbb{Z})^\times$, $g = \overline{5}$ *and* $h = \overline{18}$. *Then:*

k	1	2	3	4	5	6	7	...
g^k	$\overline{5}$	$\overline{25}$	$\overline{14}$	$\overline{33}$	$\overline{17}$	$\overline{11}$	$\overline{18}$	...

and so $\text{dlog}_g(h) = \text{dlog}_{\overline{5}}(\overline{18}) = 7 + 36\mathbb{Z}$.

As with trial division, this method is clearly hopeless if the number of possibilities is large, i.e. if $|G|$ is large. Clearly we need a better method!

6.2 Baby-Step Giant-Step

The rough idea of the Baby-Step Giant-Step algorithm (due to **Shanks**) is to instead compute two unrelated lists of powers of g and look for a collision, i.e. a repeated entry in both. Rearranging then gives us a solution to the Discrete Log Problem.

How does it work? Well, since the discrete log in G is determined mod $|G|$, we know that there is some $0 \leq k \leq |G|$ such that $g^k = h$. Shanks' idea was to try to split k up into two pieces.

We define $m = \lceil \sqrt{|G|} \rceil$, the smallest integer greater than $\sqrt{|G|}$. Then we can write $k = i + jm$ for some $0 \leq i, j \leq m - 1$, and so $h = g^k = g^{i+jm} = g^i \cdot g^{jm}$. Rearranging gives $hg^{-jm} = g^i$, which gives us a good idea of what we should take to be our two lists!

Baby-Step Giant-Step:

1. Calculate $m = \lceil \sqrt{|G|} \rceil$ and compute the list g^i for $0 \leq i \leq m - 1$.

2. Compute $x = g^{-m}$ and compute the list hx^j for $0 \leq j \leq m-1$.

3. Find the collision between the two lists, i.e. find $0 \leq i, j \leq m-1$ such that $hx^j = g^i$. Then $h = g^{i+jm}$, giving $k = i + jm$ as a representative for $\mathrm{dlog}_g(h)$.

Example 6.2. *Let's take* $G = (\mathbb{Z}/139\mathbb{Z})^\times$, $g = \overline{2}$ *and* $h = \overline{112}$. *Then* $m = \lceil\sqrt{138}\rceil = 12$ *and the first list is as follows:*

i	0	1	2	3	4	5	6	7	8	9	10	11
g^i	$\overline{1}$	$\overline{2}$	$\overline{4}$	$\overline{8}$	$\overline{16}$	$\overline{32}$	$\overline{64}$	$\overline{128}$	$\overline{117}$	$\overline{95}$	$\overline{51}$	$\overline{102}$

We now compute $x \equiv g^{-m} \equiv 2^{-12} \equiv 65^{-1} \equiv 77 \bmod 139$ *(using say Euclid's Algorithm), and so the second list begins as follows:*

j	0	1	2	3	4	...
hx^j	$\overline{112}$	$\overline{6}$	$\overline{45}$	$\overline{129}$	$\overline{64}$	...

We notice the collision $g^6 = \overline{64} = hx^4$, *so that* $h = g^{6+4\cdot 12} = g^{54}$. *This tells us that* $k = 54$ *is a possible discrete log, i.e.* $\mathrm{dlog}_g(h) = 54 + 138\mathbb{Z}$.

We won't go into much detail on the performance of this algorithm, but note that this algorithm is much better than brute force. In terms of complexity, Baby-Step Giant-Step takes $O(\sqrt{|G|})$ time, as opposed to the $O(|G|)$ time that brute force took.

6.3 The Pollard Rho method for discrete logs

Recall that the idea behind the Pollard Rho method of factorisation was to iterate a random enough function enough times to detect a collision mod p, hopefully then giving a non-trivial solution to $a^2 \equiv b^2 \bmod N$.

It turns out that a similar method works for the Discrete Log Problem, but it's a bit more complicated. We'll study the special case of $G = (\mathbb{Z}/p\mathbb{Z})^\times$ in this course, but there is a more general version for arbitrary finite cyclic group G.

Suppose that $g \in (\mathbb{Z}/p\mathbb{Z})^\times$ is a generator and that $h = g^k$ is an instance of the Discrete Log Problem. The idea is to search for a collision of the form $g^a h^b = g^c h^d$, since this implies that:

$$g^a h^b = g^a (g^k)^b = g^{a+bk} = g^{c+dk} = g^c (g^k)^d = g^c h^d.$$

Since the powers of g repeat every $p-1$ we would then know that

$$a + bk \equiv c + dk \bmod p-1 \Leftrightarrow (b-d)k \equiv (c-a) \bmod p-1.$$

If we find that $b-d$ is coprime with $p-1$ then we could solve this congruence and conclude that $\mathrm{dlog}_g(h) = k + (p-1)\mathbb{Z}$. (If not then cancelling $t = \gcd(b-d, p-1)$ throughout determines k mod $\frac{p-1}{t}$, which might still lead to a small number of possibilities for $k \bmod p-1$).

The question is now how we can find such a collision...

As earlier, brute force is always a solution. We could simply compute a table of $g^a h^b$ values for increasing a, b values until we observe a collision. However, the power of the previous method was to use a suitably random function to get to the collision.

Pollard's idea was to navigate the list of values $g^a h^b$ in a much more random way. Then we might expect to converge on a collision much faster.

There are three simple ways to modify x:

$$\begin{aligned} x &\mapsto gx & (a,b) &\mapsto (a+1,b) \bmod p-1 \\ x &\mapsto x^2 & (a,b) &\mapsto (2a,2b) \bmod p-1 \\ x &\mapsto hx & (a,b) &\mapsto (a,b+1) \bmod p-1 \end{aligned}$$

and so what we could do is start with an initial $x_0 = g^{a_0} h^{b_0} \in G$ and choose a random sequence of these three maps to do, looking out for a collision along the way. However, as we've discovered, humans are bad at randomness...so we do something a bit more methodical.

Pollard suggested dividing the set $\{1, 2, ..., p-1\}$ into three equal-ish pieces and then assigning one of the transformations to each (so that the output depends on the input). The easiest way to do this is to simply split the interval into thirds and define a function as follows:

$$f(x) = \begin{cases} gx & \text{if } 1 \leq x < \frac{p}{3} \\ x^2 & \text{if } \frac{p}{3} < x < \frac{2p}{3} \\ hx & \text{if } \frac{2p}{3} < x \leq p-1 \end{cases}$$

This gives the following algorithm:

Pollard Rho for discrete logs:

1. Choose a starting pair (a_0, b_0) of integers $1 \leq a_0, b_0 \leq p-1$ and compute the starting input $x_0 = g^{a_0} h^{b_0}$.
2. Iterate the function $f(x)$ to get two lists, $x_0, x_1, x_2, ...$ and $(a_0, b_0), (a_1, b_1), (a_2, b_2), ...$
3. Check for a collision $x_i = x_j$ as you go. If you find one then try to solve the congruence $(b_i - b_j)k \equiv (a_j - a_i) \bmod p-1$. If a unique solution exists then $\text{dlog}_g(h) = k + (p-1)\mathbb{Z}$. Otherwise, solve to find $k \bmod \frac{p-1}{t}$ where $t = \gcd(b_i - b_j, p-1)$ and brute force $k \bmod p-1$.

Example 6.3. *Let's take $G = (\mathbb{Z}/97\mathbb{Z})^\times$, $g = \overline{5}$ and $h = \overline{69}$. Then the table is as follows (assuming starting pair is $(a_0, b_0) = (1, 1)$:*

i	0	1	2	3	4	5	6	7	8	9
x_i	$\overline{54}$	$\overline{6}$	$\overline{30}$	$\overline{53}$	$\overline{93}$	$\overline{15}$	$\overline{75}$	$\overline{34}$	$\overline{89}$	$\overline{30}$
a_i	1	2	3	4	8	8	9	9	18	18
b_i	1	2	2	2	4	5	5	6	12	13

We find the collision $x_2 = x_9$ and we now solve the congruence $(b_2 - b_9)k \equiv (a_9 - a_2) \bmod 96$, i.e. $-11k \equiv 15 \bmod 96$. The solution is given by $k \equiv 51 \bmod 96$ and so $dlog_g(h) = 51 + 96\mathbb{Z}$.

6.4 Index Calculus

So far, we've seen a couple of simple ways to attack the Discrete Log Problem for arbitrary cyclic groups. We might ask whether there is an algorithm that is analogous to Dixon's algorithm for factorisation. It turns out that there is, but we must once again restrict to the groups $(\mathbb{Z}/p\mathbb{Z})^\times$ for p prime.

The reason for this is that in order to turn the factorisation problem into a Linear Algebra problem, we needed to use prime factorisation to represent elements of $\mathbb{Q}^\times$ as vectors. For elements of a general cyclic group G there is no obvious analogue of this concept (e.g. what is the prime factorization of a matrix, a symmetry of a shape, a permutation?).

Index calculus is a method of using Linear Algebra to solve the Discrete Log Problem in $(\mathbb{Z}/p\mathbb{Z})^\times$.

Here's the rough idea. Suppose that $G = (\mathbb{Z}/p\mathbb{Z})^\times$ with p prime, and $h = g^k$ for some generator $g \in G$ and $k \in \mathbb{Z}$. If we factorise h into prime powers then we get that:

$$h \equiv \prod_i p_i^{e_i} \bmod p.$$

It follows that:

$$k \equiv \mathrm{dlog}_g(h) \equiv \mathrm{dlog}_g\left(\prod_i p_i^{e_i}\right) \equiv \sum_i e_i \cdot \mathrm{dlog}_g(p_i) \bmod p-1.$$

If we knew the values $\mathrm{dlog}_g(p_i)$ for each $p_i \mid h$ then we would be done. Unfortunately, we haven't got much control over which primes can appear.

Idea: Let's try and use smoothness again!

Choose a smoothness bound $B \geq 2$. If we can modify h in some nice way to force a B-smooth factorisation, then we might only need to know the values $\mathrm{dlog}_g(p_i)$ for each prime $1 \leq i \leq \pi(B)$. The task of finding this fixed list of dlogs might then be manageable!

A good way to modify h is as follows. Suppose that we find $s \in \mathbb{Z}$ such that:

$$hg^s \equiv \prod_{1 \leq i \leq \pi(B)} p_i^{e_i} \bmod p.$$

Then it follows that:

$$\mathrm{dlog}_g(hg^s) \equiv \mathrm{dlog}_g(h) + \mathrm{dlog}_g(g^s) \equiv k + s \equiv \mathrm{dlog}_g\left(\prod_{1 \leq i \leq \pi(B)} p_i^{e_i}\right) \equiv \sum_{1 \leq i \leq \pi(B)} e_i \cdot \mathrm{dlog}_g(p_i) \bmod p-1.$$

Rearranging gives:

$$\mathrm{dlog}_g(h) \equiv k \equiv \left(\sum_{1 \leq i \leq \pi(B)} e_i \cdot \mathrm{dlog}_g(p_i)\right) - s \bmod p-1.$$

Great...but now we wonder how we're ever going to find the values $\mathrm{dlog}_g(p_i)$ for $1 \leq i \leq \pi(B)$. The best way is to use Linear Algebra!

Suppose that we compute a bunch of random powers g^{a_m} and find that

$$g^{a_m} \equiv \prod_{1 \leq i \leq \pi(B)} p_i^{e_{m,i}} \bmod p,$$

is B-smooth. Then for each power we learn a congruence:

$$a_m \equiv \mathrm{dlog}_g \left(\prod_{1 \leq i \leq \pi(B)} p_i^{e_{m,i}} \right) \equiv \sum_{1 \leq i \leq \pi(B)} e_{m,i} \cdot \mathrm{dlog}_g(p_i) \bmod p-1.$$

If we gather enough of these congruences then maybe we can solve and determine the values $\mathrm{dlog}_g(p_i)$ for $1 \leq i \leq \pi(B)$.

Recall the following map from the Dixon's method section:

$$\begin{aligned} \psi : \mathbb{Q}^\times &\longrightarrow \bigoplus_{n \geq 1} \mathbb{Z} \\ n = \prod_i p_i^{e_i} &\longmapsto (e_1, e_2, e_3, ...). \end{aligned}$$

As in that section, we define the vector $\mathbf{e}_m = \psi\left(\prod_i p_i^{e_{m,i}}\right) = (e_{m,1}, e_{m,2}, e_{m,3}, ...) \in \bigoplus_{n \geq 1} \mathbb{Z}$. This time we'll denote by

$$\overline{\mathbf{e}}_m = (\overline{e}_{m,1}, \overline{e}_{m,2}, \overline{e}_{m,3}, ...) \in \bigoplus_{n \geq 1} \mathbb{Z}/(p-1)\mathbb{Z}$$

the mod $p-1$ reduction of this vector.

The thing to notice is that the relation:

$$a_m \equiv \sum_{1 \leq i \leq \pi(B)} e_{m,i} \cdot \mathrm{dlog}_g(p_i) \bmod p-1,$$

can be written as:

$$\overline{e}_m \cdot \mathbf{d} \equiv a_m \bmod p-1,$$

where $\mathbf{d} = (\mathrm{dlog}_g(2), \mathrm{dlog}_g(3), \mathrm{dlog}_g(5), ...)$. The entire collection of relations gathered then translates to the matrix equation $E\mathbf{d} \equiv \mathbf{a} \bmod p-1$, where E is the mod $p-1$ matrix:

$$\begin{pmatrix} \overline{e_{1,1}} & \overline{e_{1,2}} & \overline{e_{1,3}} & ... & \overline{e_{1,k}} \\ \overline{e_{2,1}} & \overline{e_{2,2}} & \overline{e_{2,3}} & ... & \overline{e_{2,k}} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \overline{e_{\pi(B),1}} & \overline{e_{\pi(B),2}} & \overline{e_{\pi(B),3}} & ... & \overline{e_{\pi(B),k}} \end{pmatrix},$$

and $\mathbf{a} = (a_1, a_2, a_3, ..., a_k) \in \mathbb{Z}^k$.

If we can solve the above matrix equation then we recover the vector $\mathbf{d}$ and hence learn all of the dlogs of small primes!

Let's collect together our findings and turn it into an algorithm.

Index Calculus:

Step 1 (Find dlogs of small primes):

1. Choose a smoothness bound $B \geq 2$ and list the primes $p_1, p_2, ..., p_{\pi(B)} \leq B$.
2. Search for integers $1 \leq a_1, a_2, ..., a_k \leq p-1$ such that:
$$g^{a_m} \equiv \prod_{1 \leq i \leq \pi(B)} p_i^{e_{m,i}} \bmod p,$$
i.e. with RHS being B-smooth.
3. Form the matrix E whose rows are the first $\pi(B)$ entries of the vectors $\overline{e}_m$, i.e. the mod $p-1$ matrix:
$$\begin{pmatrix} \overline{e_{1,1}} & \overline{e_{1,2}} & \overline{e_{1,3}} & ... & \overline{e_{1,k}} \\ \overline{e_{2,1}} & \overline{e_{2,2}} & \overline{e_{2,3}} & ... & \overline{e_{2,k}} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \overline{e_{\pi(B),1}} & \overline{e_{\pi(B),2}} & \overline{e_{\pi(B),3}} & ... & \overline{e_{\pi(B),k}} \end{pmatrix}.$$
4. Solve the matrix equation $E\mathbf{v} \equiv \mathbf{a} \bmod p-1$, where $\mathbf{a} = (a_1, a_2, ..., a_k) \in \mathbb{Z}^k$. If there's no solution then generate more values a_i.
5. Then $\mathbf{v} \equiv \mathbf{d} \equiv (\text{dlog}_g(2), \text{dlog}_g(3), ..., \text{dlog}_g(p_{\pi(B)})) \bmod p-1$.

Step 2 (Relate dlog of h to dlogs of small primes):

1. Find $s \in \mathbb{Z}$ such that:
$$hg^s \equiv \prod_{1 \leq i \leq \pi(B)} p_i^{e_i} \bmod p,$$
i.e. RHS is B-smooth.
2. Then compute that $\text{dlog}_g(h) \equiv \left(\sum_{1 \leq i \leq \pi(B)} e_i \cdot \text{dlog}_g(p_i)\right) - s \bmod p-1$, using the known dlogs from Step 1.

Example 6.4. *Let's take* $G = (\mathbb{Z}/83\mathbb{Z})^\times$, $g = \overline{2}$, $h = \overline{31}$ *and* $B = 7$.

Step 1: *We must find* $\boldsymbol{d} = (dlog_g(2), dlog_g(3), dlog_g(5), dlog_g(7))$.

We run through the powers of g *mod* p*:*

$$\begin{aligned}
2^1 &\equiv 2 \bmod 83 \\
2^2 &\equiv 2^2 \bmod 83 \\
&\vdots \\
2^7 &\equiv 128 \equiv 45 \equiv 3^2 \cdot 5 \bmod 83 \\
2^8 &\equiv 7 \bmod 83 \\
2^9 &\equiv 14 \equiv 2 \cdot 7 \bmod 83 \\
&\vdots \\
2^{12} &\equiv 112 \equiv 29 \bmod 83 \\
2^{13} &\equiv 2 \cdot 29 \bmod 83 \\
2^{14} &\equiv 2^2 \cdot 29 \bmod 83 \\
&\vdots \\
2^{17} &\equiv 15 \equiv 3 \cdot 5 \bmod 83 \\
&\vdots
\end{aligned}$$

Discarding the congruences such that the RHS is not 7*-smooth, we try to solve the matrix equation* $E\boldsymbol{v} \equiv \boldsymbol{a} \bmod p - 1$, *where* $\boldsymbol{a} = (1, 7, 8, 17)$ *and*

$$E = \begin{pmatrix} \overline{1} & \overline{0} & \overline{0} & \overline{0} \\ \overline{0} & \overline{2} & \overline{1} & \overline{0} \\ \overline{0} & \overline{0} & \overline{0} & \overline{1} \\ \overline{0} & \overline{1} & \overline{1} & \overline{0} \end{pmatrix}.$$

(Remember, this is a matrix with mod 82 *entries...not mod* 83*).*

We solve this system using row reduction as follows:

$$\left(\begin{array}{cccc|c} \overline{1} & \overline{0} & \overline{0} & \overline{0} & \overline{1} \\ \overline{0} & \overline{2} & \overline{1} & \overline{0} & \overline{7} \\ \overline{0} & \overline{0} & \overline{0} & \overline{1} & \overline{8} \\ \overline{0} & \overline{1} & \overline{1} & \overline{0} & \overline{17} \end{array}\right) \sim \left(\begin{array}{cccc|c} \overline{1} & \overline{0} & \overline{0} & \overline{0} & \overline{1} \\ \overline{0} & \overline{1} & \overline{0} & \overline{0} & \overline{-10} \\ \overline{0} & \overline{0} & \overline{1} & \overline{0} & \overline{27} \\ \overline{0} & \overline{0} & \overline{0} & \overline{1} & \overline{8} \end{array}\right).$$

We find that there is a unique solution $\boldsymbol{v} = (\overline{1}, \overline{-10}, \overline{27}, \overline{8})$, *so that* $\boldsymbol{d} \equiv (1, -10, 27, 8) \bmod 82$.

Step 2: *We search for $s \in \mathbb{Z}$ such that hg^s is 7-smooth:*

$$\begin{aligned} 31 \cdot 2 &\equiv 2 \cdot 31 \bmod 83 \\ 31 \cdot 2^2 &\equiv 124 \equiv 41 \bmod 83 \\ 31 \cdot 2^3 &\equiv 2 \cdot 41 \bmod 83 \\ 31 \cdot 2^4 &\equiv 81 \equiv 3^4 \bmod 83. \end{aligned}$$

It now follows that:

$$dlog_g(h) \equiv 4 \cdot dlog_g(3) - 4 \equiv 4 \cdot (-10) - 4 \equiv -44 \equiv 38 \bmod 82,$$

so that $dlog_g(h) = 38 + 82\mathbb{Z}$, i.e. $31 \equiv 2^{38} \bmod 83$.

Exercise 6.5. *Check that the dlog we computed is correct.*

It might seem annoying that we did a lot of work in Step 1 to find the dlogs of all of the primes $2, 3, 5, 7$...only to find later that we didn't need all of these (we just needed $\text{dlog}_g(3)$ in the end). However, to compute even a single dlog value we needed to use Linear Algebra...and to do this it was actually helpful to consider a batch of dlog values at once!

The index calculus method for solving the Discrete Log Problem in $(\mathbb{Z}/p\mathbb{Z})^\times$ is very handy, but there are much better generalisations that also use Linear Algebra, e.g. Quadratic Sieve and Number Field Sieve. These methods are great and work fast for large problem sizes, but none of these methods can run fast enough to break a generic real world Diffie-Hellman/El-Gamal scheme (which is again either fortunate or unfortunate, however you view things).

7 Post Quantum Cryptography: The potential future of security

So far, we have seen many public key schemes that rely on the toughness of mathematical problems to guarantee security. For example, the RSA scheme relies on the toughness of the Semiprime Factoring Problem and the El-Gamal scheme relies on the toughness of the Discrete Log Problem for certain (presentations of) cyclic groups.

However, the world might currently be on the brink of a major advance in technology. **Quantum computers** rely on the nature of quantum mechanics to compute. In particular their base unit is not a bit (a 0 or a 1) but a **qubit** (a superposition $\alpha|0\rangle + \beta|1\rangle$ for $(\alpha, \beta) \in \mathbb{C}^2$).

The fact that a qubit has much more freedom is partly what makes quantum computers much more powerful than conventional computers. Another strong feature is that qubits can be **entangled** using the tensor product, and so a quantum computer can potentially store information on all 2^n states of a system using only n qubits.

Due to the above, quantum computers are very good at performing **Fourier transforms** and so can detect periodicity and solve such problems very quickly. In particular, **Shor's algorithm** can solve both the Semiprime Factoring Problem and the Discrete Log Problem in polynomial time on a quantum computer. When quantum computers become good enough, schemes like RSA, Diffie-Hellman and El-Gamal will become **insecure**.

There is currently a global fight for quantum supremacy! Thus, there is a scramble for quantum secure algorithms, ones that rely on problems that a quantum computer is not known to be able to solve easily. Certain **lattice** problems are thought to be such problems, but more on that later.

7.1 Knapsack schemes

Consider the following problem. Let S be a set of positive integers. If I choose a subset $S' \subseteq S$ then I can compute the **sum** of the elements of S'. This is of course easy and fast to do, but suppose I instead gave you only the sum and asked you what S' is. This turns out to be a tough problem.

The Knapsack Problem: Given an increasing sequence of positive integers $a_1 < a_2 < ... < a_n$ and a sum of distinct elements $s = \sum_{1 \leq t \leq k} a_{i_t}$ for some $1 \leq i_k < i_{k-1} < ... < i_1 \leq n$, determine the set $S = \{a_{i_1}, a_{i_2}, ..., a_{i_k}\}$.

This is a very difficult problem to solve, even though it sounds easy. The basic reason for this is that if $|S| = n$ is a generic set of positive integers then there are 2^n possible sums of distinct elements. This is **huge** and infeasible to search through for **large** n.

Ok, so we've found a hard problem that could be used in cryptography. But remember, we need to find an easy version of the problem for Bob to solve.

Thankfully there are choices of sequence that are easy to solve.

Definition 7.1. *A sequence $a_1 < a_2 < ... < a_n$ of positive integers is* ***superincreasing*** *if $a_{m+1} > \sum_{1 \leq t \leq m} a_t$ for any $1 \leq m \leq n-1$.*

Example 7.2. *The sequence $a_n = 2^{n-1}$ is superincreasing since for any $1 \leq m \leq n$ we have:*

$$\sum_{1 \leq t \leq m} a_t = \sum_{1 \leq t \leq m} 2^{t-1} = 2^m - 1 < 2^m = a_{m+1}.$$

Similarly, the sequence $a_n = b^{n-1}$ is superincreasing for any $b \geq 2$.

The sequence $1, 4, 6, 20, 34, 70$ is superincreasing, but the sequence $1, 4, 6, 20, 30, 70$ is not (since $1 + 4 + 6 + 20 = 31 > 30$).

Theorem 7.3. *For superincreasing sequences, there is an algorithm that solves the Knapsack Problem in polynomial time.*

Proof. We use a greedy algorithm. Choose the biggest term a_{i_1} in the sequence that is less than or equal to s. We then choose the biggest term a_{i_2} in the sequence that is less than or equal to $s - a_{i_1}$. Do the same for $s - a_{i_1} - a_{i_2}$. Keep going until reaching a term less than every element of the sequence. Then we claim that $s = a_{i_1} + a_{i_2} + ... + a_{i_h}$ solves the Knapsack Problem.

Suppose that the true solution is $s = a_{j_1} + a_{j_2} + ... + a_{j_k}$ with $j_k < j_{k-1} < ... < j_1$. We prove that $a_{i_1} = a_{j_1}$. It then follows that $a_{i_2} = a_{j_2}$, by considering the same Knapsack problem but with $s - a_{j_1}$. Continuing recursively it then follows that $a_{i_t} = a_{j_t}$ for all $1 \leq t \leq k$ (which forces $h = k$ too). Thus, the true solution would match the output of the algorithm.

We split into two cases:

- If $a_{i_1} < a_{j_1}$ then the algorithm underestimates s, since:

$$s \geq a_{j_1} > a_{j_{1-1}} + ... + a_1 \geq a_{i_1} + ... + a_1 \geq a_{i_1} + a_{i_2} + ... + a_{i_h}.$$

- If $a_{i_1} > a_{j_1}$ then the algorithm overestimates s, since

$$s \leq a_{j_1} + ... + a_1 < a_{j_1+1} \leq a_{i_1} \leq a_{i_1} + a_{i_2} + ... + a_{i_h}.$$

Thus $a_{i_1} = a_{j_1}$ as expected. □

Example 7.4. *You already know an example of this algorithm. We showed above that $a_n = 2^{n-1}$ is a superincreasing sequence, and the Knapsack Problem for this sequence is equivalent to writing numbers in* ***binary****. The above algorithm is the usual way that you do this.*

For example, take sequence $1, 2, 4, 8, 16, 32$ and $s = 22$. The biggest element that is less than or equal to s is 16. The biggest element less than or equal to $s - 16 = 6$ is 4. The biggest element less than or equal to $s - 16 - 4 = 2$ is 2. Thus $s = 2 + 4 + 16$ solves this Knapsack Problem (i.e. the number 23 in binary is 10110).

Example 7.5. *For a random example, take sequence $3, 5, 9, 20, 41$ and $s = 53$. This sequence is superincreasing. The biggest element less than or equal to s is 41. The biggest element less than or equal to $s - 41 = 12$ is 9. The biggest element less than or equal to $s - 41 - 9 = 3$ is 3. Thus $s = 3 + 9 + 41$ solves this Knapsack Problem.*

The above algorithm can fail for a non-superincreasing sequence, e.g. for sequence $2, 3, 4$ and $s = 5$ the biggest element less than or equal to s is 4, but then $s - 4 = 1$ is smaller than everything in S.

We are almost ready to see the Knapsack scheme. We just have to figure out how Bob can turn the easy version of the Knapsack problem into a hard version. He can start with a superincreasing sequence, choose a modulus M, a positive integer w coprime with M, and compute the new sequence $wa_1 < wa_2 < ... < wa_n$. Reducing this sequence mod M produces a **random** looking sequence that is not likely to be superincreasing (it probably isn't even increasing).

The Knapsack scheme:

1. Bob chooses a superincreasing sequence $a_1 < a_2 < ... < a_n$, a modulus $M > \sum_{1 \leq t \leq n} a_t$ and a number w coprime with M. His public key is the sequence $b_1, b_2, ..., b_n$, formed by reducing the sequence $wa_1 < wa_2 < ... < wa_n \bmod M$. His private key is the pair (M, w).
2. To send a message m to Bob, Alice converts it to an n long bit string $x = (x_1, x_2, ..., x_n)$, in her favourite way, and sends $c = \sum_{1 \leq t \leq n} x_t b_t$.
3. Bob decrypts by computing $c' \equiv w^{-1}c \bmod M$ and solving the Knapsack Problem for the superincreasing sequence $a_1 < a_2 < ... < a_n$ and $s = c'$.

Lemma 7.6. *The above protocol allows Bob to read the message.*

Proof. Bob receives $c = \sum_{1 \leq t \leq n} x_t b_t$ and the claim is that x also solves the Knapsack Problem for the sequence $a_1 < a_2 < ... < a_n$ and $s = c'$. This is clear since:

$$c' \equiv w^{-1}c \equiv w^{-1} \sum_{1 \leq t \leq n} x_t b_t \equiv \sum_{1 \leq t \leq n} x_t (w^{-1} b_t) \equiv \sum_{1 \leq t \leq n} x_t a_t \bmod M.$$

By the condition $M > \sum_{1 \leq t \leq n} x_t a_t$ we have equality $c' = \sum_{1 \leq t \leq n} x_t a_t$, proving our claim. □

Example 7.7. *Bob chooses the superincreasing sequence* $1, 2, 4, 8, 16, 32$. *Using modulus* $M = 65$ *and* $w = 11$ *this becomes the sequence* $11, 22, 44, 23, 46, 27$.

Alice wishes to send the bit string 100110 *to Bob. She sends* $c = 11 + 23 + 46 = \mathbf{80}$ *to Bob.*

Bob computes $w^{-1} \equiv 6 \bmod M$ *and computes* $c' \equiv 6 \cdot 80 \equiv \mathbf{25} \bmod 65$. *Solving the Knapsack Problem for the superincreasing sequence and* $s = 25$ *gives* 100110 *as expected.*

We say a little more about the security of this scheme. Given a generic Knapsack Problem, for a sequence $b_1, b_2, ..., b_n$ and a target $s = \sum_{1 \leq t \leq n} x_t b_t$, consider the matrix:

$$\begin{pmatrix} 2 & 0 & 0 & ... & b_1 \\ 0 & 2 & 0 & ... & b_2 \\ 0 & 0 & 2 & ... & b_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & 1 & ... & s \end{pmatrix}.$$

Let $\mathbf{r}_1, \mathbf{r}_2, ..., \mathbf{r}_{n+1}$ denote the rows of this matrix.

Since $s = \sum_{1 \leq t \leq n} x_t b_t$ we note that the vector

$$\mathbf{v} = \left(\sum_{1 \leq t \leq n} x_t \mathbf{r}_t \right) - \mathbf{r}_{n+1} = (2x_1 - 1, 2x_2 - 1, ..., 2x_n - 1, 0)$$

is an integer linear combination of the vectors $\mathbf{r}_i$. The set of vectors:

$$\mathcal{L} = \{\alpha_1 \mathbf{r}_1 + \alpha_2 \mathbf{r}_2 + ... + \alpha_{n+1} \mathbf{r}_{n+1} \,|\, \alpha_i \in \mathbb{Z}\},$$

is an example of a structure called a **lattice** (more on these later!).

Notice that since each $x_i \in \{0, 1\}$, the vector $\mathbf{v}$ has quite a short Euclidean length

$$||\mathbf{v}|| = \sqrt{(2x_1 - 1)^2 + (2x_2 - 1)^2 + ... + (2x_n - 1)^2} = \sqrt{n}.$$

For reasons which we'll see soon, this is likely to be the **shortest** non-zero vector in $\mathcal{L}$, and if n is large enough then this vector is very hard to find.

This problem is considered a very difficult problem to solve in large enough dimensions, even with a quantum computer. However, there exist lattice reduction algorithms that can approximately solve such problems for lattices of low enough rank.

Sadly, in order to use the Knapsack scheme with a reasonable key size we would like $\mathcal{L}$ to have rank $n + 1 > 300$, but for these values of n lattice reduction algorithms are likely to produce the shortest vector $\mathbf{v}$ that solves the corresponding Knapsack Problem.

While Knapsack schemes are known to be insecure for key sizes we might care about, they at least hint that lattice problems might be of use in cryptography. Indeed, this is the case and there are a great deal of lattice based schemes that have been proposed over the last few decades. The current favourite is the **NTRU** scheme, which uses the toughness of the **Shortest Vector Problem** and the **Closest Vector Problem** for lattices. We'll see this soon.

7.2 A simple noisy scheme

The reason that quantum computers are so good at solving the Semiprime Factorisation Problem and the Discrete Log Problem is the fact that both of these problems can be related to the problem of finding the period of a certain map. The Quantum Fourier Transform is able to compute this period in polynomial time on a quantum computer.

How can we design schemes that are not susceptible to such attacks? The best idea that we currently have is to introduce **random noise** into the system, thus destroying the periodicity.

Let's see an example of good/bad ways to do this.

Example 7.8. *If I give you an integer $n \in \mathbb{Z}$ and promise you that it's a multiple of $s \in \mathbb{Z}$ then $n = as$ for some $a \in \mathbb{Z}$ and you know how to find a (you compute $\frac{n}{s} = a$). This is a periodic problem...the multiples of s repeat every s.*

*Let's add some noise! Suppose I give you an integer $n \in \mathbb{Z}$ and promise you that it's close to a multiple of $s \in \mathbb{Z}$ then $n = as + r$ for some small **noise** $r \in \mathbb{Z}$. We would know how to find a and r by calculating $\frac{n}{s} = a + \frac{r}{s}$, rounding to get a and then rearranging to get $r = n - as$.*

For example, if I give you $n = 168$ and promise you that it's close to a multiple of $s = 11$ then you would be able to calculate $\frac{n}{s} = \frac{168}{11} = 15 + \frac{3}{11}$ and know that it's within $r = 3$ of $15 \cdot 11$.

*Ok, so we added some noise to an integer problem and it was still easy...this is clearly a **bad** thing to do!*

Example 7.9. *It turns out that a **good** thing to do is to consider this problem mod a prime p.*

If I give you an integer $n \in \mathbb{Z}$ and promise you that it's a multiple of $s \bmod p$, then $n \equiv as \bmod p$ for some $a \in \mathbb{Z}$ and you know how to find a (you solve the linear congruence using your favourite technique). This is again a periodic problem...the multiples of $s \bmod p$ repeat every s.

*Let's add some noise! Suppose I give you an integer $n \in \mathbb{Z}$ and promise you that it's close to a multiple of $s \bmod p$ then $n \equiv as + r \bmod p$ for some small **noise** $r \bmod p$. Well...this problem is also easy, since I can take $r = 0$ and solve the congruence $n \equiv as \bmod p$ (any two non-zero elements of $(\mathbb{Z}/p\mathbb{Z})^\times$ are multiples of each other, since it's a field!)*

However, now suppose that I choose the multiple and the noise to both be small. There is then no obvious way to solve this congruence, since the numbers $as \bmod p$ can vary randomly between 0 and $p - 1$ and make a smaller random subset...unlike the problem in $\mathbb{Z}$, there is no good notion of size mod p that can be used to solve this problem!

For example, suppose that I give you $n \equiv 168 \bmod 199$ and promise you that it's within 3 of a small multiple of $s \equiv 134 \bmod 199$. The table of multiples of $s \bmod p$ is quite erratic:

a	1	2	3	4	5	6	7	8	9	10	...
$as \bmod p$	134	69	4	138	73	8	142	77	12	146	...

It is not immediately obvious that $a = 25$ and $r = -2$, i.e. $25 \cdot 134 \equiv 166 \bmod 199$ is 2 away from $168 \bmod 199$.

We've found what seems to be a **hard** problem! But for this to be useful to Cryptography we need to find an **easy** version of the problem that Alice can use as her trapdoor.

Example 7.10. *If instead I challenge you to find a small multiple of $s \equiv 11 \bmod 199$ that is within 3 of a multiple of $n \equiv 168 \bmod 199$, then we notice that this is the same problem as the integer problem above. Since $168 = 15 \cdot 11 + 3$, we immediately find that $a = 15$ and $r = 3$ works.*

This was an easy problem because s was itself small and so the list of multiples of $s \bmod p$ takes a while to reach size p, so the randomness takes a while to come into effect):

a	1	2	3	4	5	6	7	8	9	10	...	15	...	18	19	...
$as \bmod p$	11	22	33	44	55	66	77	88	99	110	...	165	...	198	10	...

Ok, so Alice should choose a small s to use as part of her private key (but not too small!). She then has to somehow transform this to a random looking number mod p to use as part of her public key. A good way to do this is to compute $v \equiv st^{-1} \bmod p$ for some small-ish t (inverses of small

elements are unlikely to be small mod p). The following scheme makes precise what each instance of the word "small" should be.

A simple noisy scheme:

1. Alice chooses a large prime p and coprime integers $\sqrt{\frac{p}{4}} < s < \sqrt{\frac{p}{2}}$ and $1 < t < \sqrt{\frac{p}{2}}$. The pair (s, t) is her private key.
2. Alice computes $v \equiv st^{-1} \bmod p$ and releases (p, v) as her public key.
3. Bob encrypts a message m to Alice by converting it to a small integer noise value $r < \sqrt{\frac{p}{4}}$ and sending Alice the value $e \equiv av + r \bmod p$ for some random choice of integer $1 < a < \sqrt{\frac{p}{2}}$.
4. Alice decrypts by computing $b \equiv te \bmod p$ (with $0 < b < p$), then computes $r \equiv t^{-1}b \bmod s$.

Proposition 7.11. *The above protocol allows Alice to retrieve the message.*

Proof. Alice first computes:

$$b \equiv te \equiv t(av + r) \equiv t(ast^{-1} + r) \equiv as + tr \bmod p.$$

Note that $0 < b < p$ and that:

$$0 < as + tr < \sqrt{\frac{p}{2}}\sqrt{\frac{p}{2}} + \sqrt{\frac{p}{2}}\sqrt{\frac{p}{4}} = \frac{p}{2} + \frac{p}{2\sqrt{2}} < p.$$

Since these integers are congruent mod p and are both between 1 and $p - 1$, it must be that $b = as + tr$ as integers.

Alice then finds that:

$$t^{-1}b \equiv t^{-1}(as + tr) \equiv ast^{-1} + r \equiv r \bmod s$$

□

Example 7.12. *Alice chooses* $p = 137$ *and computes that* $\sqrt{\frac{p}{4}} \approx 5.85$ *and* $\sqrt{\frac{p}{2}} \approx 8.28$. *She makes the valid choice* $(s, t) = (7, 3)$ *for her private key.*

Since $3^{-1} \equiv 46 \bmod 137$, *she computes that*

$$v \equiv st^{-1} \equiv 7 \cdot 3^{-1} \equiv \mathbf{48} \bmod 137$$

her public key is $(p, v) = (137, 48)$.

Bob wishes to send a message m, *which he has converted to the small noise value* $r = 2$. *He chooses* $a = 5$ *and sends*

$$e \equiv av + r \equiv 5 \cdot 48 + 2 \equiv \mathbf{105} \bmod 137$$

to Alice.

Alice decrypts by computing first that

$$b \equiv te \equiv 3 \cdot 105 \equiv \mathbf{41} \bmod 137,$$

and then learns that

$$r \equiv t^{-1}b \equiv 5 \cdot 41 \equiv \mathbf{2} \bmod 7,$$

as expected (note that we needed a different modular inverse here, $t^{-1} \equiv 5 \bmod 7$*).*

What must Eve do to break this scheme? Well she only sees the values (p, v, e) and must solve the seemingly **hard** problem we mentioned above, to find a solution to $e \equiv av + r \bmod p$ where $v \bmod p$ is **large** and a, r are small-ish.

Alice is able to recover the message since her ability to compute $b \equiv te \bmod p$ transforms the hard problem into the **easy** one, to find a solution to $b \equiv as + tr \bmod p$ where $s \bmod p$ is **small** and $a, tr \bmod p$ are small-ish.

However, before we all celebrate the invention of our new seemingly quantum secure scheme...I should point out that it's actually **highly insecure**! We'll see the full reason why in the next subsection, but for now let's translate it into another lattice problem.

The issue is that Eve can break the scheme if she can find **any** pair of coprime integers (S, T) with $1 < S, T \leq \sqrt{\frac{p}{2}}$ and $v \equiv ST^{-1} \bmod p$. Equivalently, we need to find such a solution to $Tv \equiv S \bmod p$, so that $Tv = S + Rp$ for some $R \in \mathbb{Z}$.

It then follows that the vector $\mathbf{x} = (S, T) \in \mathbb{Z}^2$ satisfies:

$$\mathbf{x} = (S, T) = (Tv - Rp, T) = T(v, 1) - R(p, 0),$$

and so is an integer linear combination of the vectors $\mathbf{r}_1 = (v, 1)$ and $\mathbf{r}_2 = (p, 0)$ of Euclidean length around p. Not only this, but $\mathbf{x}$ is a vector that has quite short Euclidean length in comparison:

$$||\mathbf{x}|| = \sqrt{S^2 + T^2} \leq \sqrt{\frac{p}{2} + \frac{p}{2}} = \sqrt{p}.$$

The set of vectors:

$$\mathcal{L} = \{\alpha_1 \mathbf{r}_1 + \alpha_2 \mathbf{r}_2 \,|\, \alpha_1, \alpha_2 \in \mathbb{Z}\},$$

is yet another example of a **lattice**, and we are seeking a vector in here that is short (of size around $\sqrt{p}$ or less).

As we'll see next lecture, finding short vectors in 2-dimensional lattices is very easy...so the scheme can easily be broken!

7.3 Lattices and Gauss reduction

We've now seen two cryptographic schemes that have their security being based on the ability to find small vectors in certain integer structures, which we called "lattices". I think it's time to understand a bit more about what lattices are and what's known about this problem!

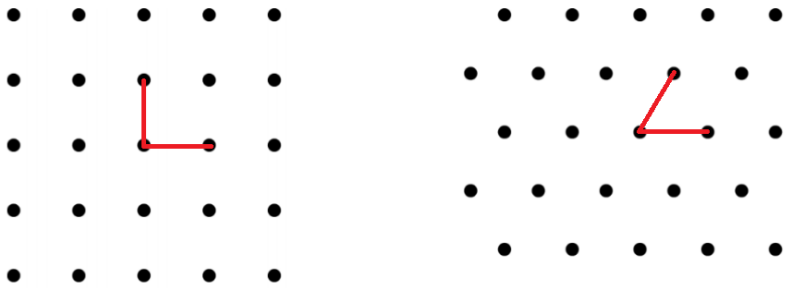
Definition 7.13. *Let $\boldsymbol{v}_1, \boldsymbol{v}_2, ..., \boldsymbol{v}_n$ be a basis for $\mathbb{R}^n$. The **lattice** spanned by this basis is the set:*

$$\mathcal{L} = \{\alpha_1 \boldsymbol{v}_1 + \alpha_2 \boldsymbol{v}_2 + ... + \alpha_n \boldsymbol{v}_n \,|\, \alpha_i \in \mathbb{Z}\},$$

i.e. the $\mathbb{Z}$-span of the basis.

In low dimensions it's possible to visualise a lattice as a regularly spaced set of points in $\mathbb{R}^n$.

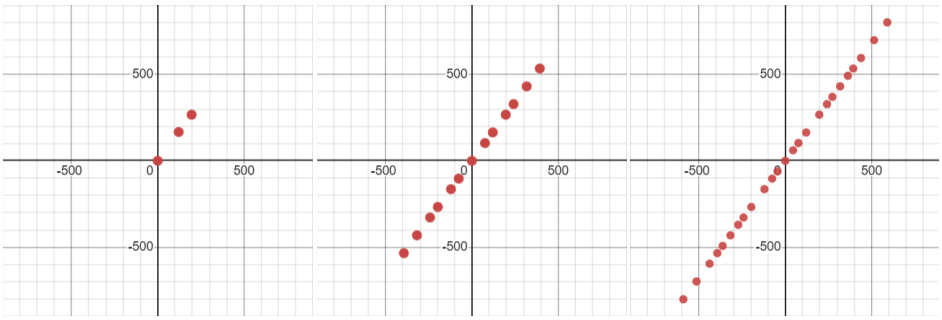
Example 7.14. *The lattice $\mathcal{L}_1$ spanned by $\boldsymbol{v}_1 = (1, 0)$ and $\boldsymbol{v}_2 = (0, 1)$ is the* ***square lattice*** *(left). The lattice $\mathcal{L}_2$ spanned by $\boldsymbol{v}_1 = (1, 0)$ and $\boldsymbol{v}_2 = (\frac{1}{2}, \frac{\sqrt{3}}{2})$ is the* ***hexagonal lattice*** *(right).*



Lattices appear throughout modern Mathematics, Physics, Chemistry and beyond. For example, a lot of highly structured compounds/crystals have their atomic structure determined by points on a lattice (e.g. diamond, graphene).

However, notice that in the above we could easily draw the picture since the vectors given were quite simple. For an arbitrary basis it might be quite difficult to understand what the arrangement of lattice points could look like.

Example 7.15. *Consider the lattice $\mathcal{L}$ spanned by $\boldsymbol{v}_1 = (121, 164)$ and $\boldsymbol{v}_2 = (197, 267)$. If we randomly add/subtract a couple of these vectors we get the following picture:*



It's not becoming clear what the entire picture of lattice points should look like. However, it turns out that $\mathcal{L}$ is just the square lattice in disguise!

The above tells us that to understand a lattice well, we might want to find a "nice basis" for it, i.e. one with small, widely spread vectors. The following problem gives us a start with that.

The Short Vector Problem: Given a lattice $\mathcal{L} \subset \mathbb{R}^n$, find the shortest non-zero vector $\mathbf{v} \in \mathcal{L}$.

This turns out to be an extremely difficult problem to solve if given a generic/random basis for a lattice of high enough dimension. Such instances of the problem are widely believed to be quantum secure.

However, for $n = 2$ the problem was completely solved in polynomial time a long time ago by Gauss (for any basis of the lattice).

Suppose that we have a lattice $\mathcal{L} \subset \mathbb{R}^2$ with basis $\mathbf{v}_1, \mathbf{v}_2$. Gauss's idea was to emulate the Euclidean algorithm and subtract a multiple of the **smaller** vector from the **larger** one. Doing this over and over again should reduce the overall size of the basis vectors, until we win and find the "smallest" basis (hence the smallest vector).

Ok, so let's assume that $||\mathbf{v}_1|| \leq ||\mathbf{v}_2||$ (otherwise we can swap them). We want to replace $\mathbf{v}_2$ with a vector of the form $\mathbf{v}_2 - \alpha\mathbf{v}_1$ of smaller size.

Lemma 7.16. *Let $\boldsymbol{v}_1, \boldsymbol{v}_2 \in \mathbb{R}^2$. The value of $\alpha \in \mathbb{R}$ that minimises $||\boldsymbol{v}_2 - \alpha\boldsymbol{v}_1||$ is $\alpha = \frac{\boldsymbol{v}_1 \cdot \boldsymbol{v}_2}{||\boldsymbol{v}_1||^2}$.*

Proof. Consider the function:

$$f(\alpha) = ||\mathbf{v}_2 - \alpha\mathbf{v}_1||^2 = (\mathbf{v}_2 - \alpha\mathbf{v}_1) \cdot (\mathbf{v}_2 - \alpha\mathbf{v}_1) = ||\mathbf{v}_2||^2 + \alpha^2||\mathbf{v}_1||^2 - 2\alpha\mathbf{v}_1 \cdot \mathbf{v}_2.$$

The stationary points of $f(\alpha)$ satisfy $f'(\alpha) = 0$, i.e. $2(\alpha||\mathbf{v}_1||^2 - \mathbf{v}_1 \cdot \mathbf{v}_2) = 0$. The only solution is $\alpha = \frac{\mathbf{v}_1 \cdot \mathbf{v}_2}{||\mathbf{v}_1||^2}$.

Since $f''(\alpha) = 2||\mathbf{v}_1||^2 > 0$, this stationary point is a minimum. □

We can almost solve the Short Vector Problem in 2-dimensions. Our problem is that in general $\alpha = \frac{\mathbf{v}_1 \cdot \mathbf{v}_2}{||\mathbf{v}_1||^2} \in \mathbb{R}$, and so the vector $\mathbf{v}_2 - \alpha\mathbf{v}_1$ is unlikely to be in our lattice $\mathcal{L}$. Instead we have to round and use the integer $\alpha = \lfloor \frac{\mathbf{v}_1 \cdot \mathbf{v}_2}{||\mathbf{v}_1||^2} \rceil \in \mathbb{Z}$.

Gauss Reduction:

1. Arrange the given basis so that $||\mathbf{v}_1|| \leq ||\mathbf{v}_2||$.
2. Replace $\mathbf{v}_2$ with $\mathbf{v}_2 - \lfloor \frac{\mathbf{v}_1 \cdot \mathbf{v}_2}{||\mathbf{v}_1||^2} \rceil \mathbf{v}_1 \in \mathcal{L}$.
3. Repeat these steps until no further progress is made.

Example 7.17. *Consider the lattice $\mathcal{L} \subset \mathbb{R}^2$ from before, with basis $\boldsymbol{v}_1 = (121, 164)$ and $\boldsymbol{v}_2 = (197, 267)$. Note that these have large length.*

We apply the Gauss reduction algorithm to produce a basis of smaller length as follows:

$$\begin{aligned}
(197, 267) - 2(121, 164) &= (-45, -61) \\
(121, 164) + 3(-45, -61) &= (-14, -19) \\
(-45, -61) - 3(-14, -19) &= (-3, -4) \\
(-14, -19) - 5(-3, -4) &= (1, 1) \\
(-3, -4) + 4(1, 1) &= (1, 0) \\
(1, 1) - (1, 0) &= (0, 1)
\end{aligned}$$

The new basis for $\mathcal{L}$ is then $\boldsymbol{w}_1 = (0, 1)$ and $\boldsymbol{w}_2 = (1, 0)$, which are incredibly small (and in this case both solve the Shortest Vector Problem).

Let's prove that Gauss's algorithm actually does give us a solution to the Shortest Vector Problem in 2-dimensions:

Theorem 7.18. *Let $\mathcal{L} \subset \mathbb{R}^2$ be a lattice.*

1. *Gauss Reduction produces a basis $\boldsymbol{w}_1, \boldsymbol{w}_2$ for $\mathcal{L}$ satisfying $||\boldsymbol{w}_1|| \leq ||\boldsymbol{w}_2||$ and having angle satisfying $\frac{2\pi}{3} \leq \theta \leq \frac{4\pi}{3}$ (i.e. the vectors are close to orthogonal).*
2. *Further, $\boldsymbol{w}_1$ solves the Shortest Vector Problem for $\mathcal{L}$.*

Proof. 1. First note that $\mathbf{w}_1, \mathbf{w}_2$ is a basis for $\mathcal{L}$, since each step of the algorithm was an invertible linear transformation with integer coefficients and integer inverse.

Since the algorithm terminates we can assume that $||\mathbf{w}_1|| \leq ||\mathbf{w}_2||$ and $\frac{|\mathbf{w}_1 \cdot \mathbf{w}_2|}{||\mathbf{w}_1||^2} < \frac{1}{2}$ (otherwise we could do another step of the algorithm).

Since $|\mathbf{w}_1 \cdot \mathbf{w}_2| = ||\mathbf{w}_1|| \cdot ||\mathbf{w}_2|| \cdot |\cos(\theta)|$, we find that:

$$|\cos(\theta)| \leq \frac{||\mathbf{w}_1||}{2||\mathbf{w}_2||} \leq \frac{||\mathbf{w}_2||}{2||\mathbf{w}_2||} = \frac{1}{2}$$

and the first claim follows.

2. Let $\mathbf{w} = a_1\mathbf{w}_1 + a_2\mathbf{w}_2 \in \mathcal{L} \backslash \{\mathbf{0}\}$ be an arbitrary non-zero vector. Then:

$$\begin{aligned} ||\mathbf{w}||^2 &= ||a_1\mathbf{w}_1 + a_2\mathbf{w}_2||^2 \\ &= a_1^2||\mathbf{w}_1||^2 + 2a_1a_2\mathbf{w}_1 \cdot \mathbf{w}_2 + a_2^2||\mathbf{w}_2||^2 \\ &\geq a_1^2||\mathbf{w}_1||^2 - 2|a_1a_2| \cdot |\mathbf{w}_1 \cdot \mathbf{w}_2| + a_2^2||\mathbf{w}_2||^2 \\ &\geq a_1^2||\mathbf{w}_1||^2 - |a_1a_2| \cdot ||\mathbf{w}_1||^2 + a_2^2||\mathbf{w}_2||^2 \\ &\geq (a_1^2 - |a_1a_2| + a_2^2)||\mathbf{w}_1||^2. \end{aligned}$$

Note that $t_1^2 - t_1t_2 + t_2^2 = \frac{3}{4}t_1^2 + (\frac{1}{2}t_1 - t_2)^2 \geq 0$, with equality if and only if $t_1 = t_2 = 0$. Since $(a_1, a_2) \neq (0, 0)$ and $a_1^2 - |a_1a_2| + a_2^2 \in \mathbb{Z}$, we must have that $||\mathbf{w}||^2 \geq ||\mathbf{w}_1||^2$, so that every non-zero vector of $\mathcal{L}$ has size at most $||\mathbf{w}_1||$. This proves the claim.

□

Let's finish by returning to the "simple noisy scheme" from the last session. Recall that Alice's private key was (p, v), where p is prime and we needed to write $v \equiv ST^{-1} \bmod p$ for some small pair of integers (S, T).

We found that this problem was equivalent to solving the Short Vector Problem in the lattice $\mathcal{L}$ spanned by $\mathbf{v}_1 = (v, 1)$ and $\mathbf{v}_2 = (p, 0)$ (since (S, T) is such a short vector). We can use Gauss Reduction to do this!

Example 7.19. *In the previous section, Alice had public key $(p, v) = (137, 48)$. Let $\mathcal{L}$ be the lattice with basis $\boldsymbol{v}_1 = (48, 1)$ and $\boldsymbol{v}_2 = (137, 0)$. We use Gauss Reduction and find that:*

$$(137, 0) - 3(48, 1) = (-7, -3)$$
$$(48, 1) + 6(-7, -3) = (6, -17)$$

It follows that a smaller basis for $\mathcal{L}$ is $\boldsymbol{w}_1 = (-7, -3)$ and $\boldsymbol{w}_2 = (6, -17)$. The shortest vector in $\mathcal{L}$ is then $\boldsymbol{w}_1$, which satisfies the requirements to break the scheme. Note that $-\boldsymbol{w}_1 = (7, 3) \in \mathcal{L}$ ... this is precisely the private key that Alice used in the example!

While we have now broken this simple system, all is not lost. In the next section we'll see that this scheme motivates a better one, whose security relies on the difficulty of the Shortest Vector Problem for **higher dimensional lattices**.

7.4 NTRU

We have just seen that the "simple noisy scheme" is incredibly easy to break, since the security depends on a Short Vector Problem in a 2-dimensional lattice ... which is easily solved by Gauss Reduction. However, the idea can be generalised to make a much better scheme that by today's standards is still believed to be (quantum) secure!

The idea is to work with **higher dimensional objects**, so that the underlying lattices are also higher dimensional (making the Short Vector Problem harder to solve).

Definition 7.20. *Let p, q, N be prime. We define the following quotient rings:*

$$R = \mathbb{Z}[x]/\langle x^N - 1\rangle, \qquad R_p = (\mathbb{Z}/p\mathbb{Z})[x]/\langle x^N - 1\rangle, \qquad R_q = (\mathbb{Z}/q\mathbb{Z})[x]/\langle x^N - 1\rangle.$$

The idea of NTRU is to try and emulate the "simple noisy scheme" but with R taking the place of $\mathbb{Z}$, R_p taking the place of $\mathbb{Z}/p\mathbb{Z}$ and R_q taking the place of $\mathbb{Z}/s\mathbb{Z}$.

Note that there are obvious maps $R \longrightarrow R_p$ and $R \longrightarrow R_q$ given by reducing the coefficients mod p and mod q respectively (i.e. $[f(x)] \longmapsto \left[\overline{f(x)}\right]$).

Recall that in the simple scheme, Alice started by choosing coprime $s, t \in \mathbb{Z}$ that are small with respect to the large prime p. We can do a similar thing in R, choosing polynomials of the form:

$$[s(x)] = [s_0 + s_1 x + s_2 x^2 + ... + s_{N-1}x^{N-1}] \in R$$
$$[t(x)] = [t_0 + t_1 x + t_2 x^2 + ... + t_{N-1}x^{N-1}] \in R$$

where all of the coefficients are small, i.e. $s_i, t_i \in \{0, 1, -1\}$. Note that these polynomials are still "small" when we reduce mod p or q.

A good way to generate such polynomials is to choose an integer $d \geq 1$ and choose d of the coefficients at random to be 1 and d of the coefficients at random to be -1 (then make the rest of the coefficients 0). This is almost what we do in NTRU, but we instead let $T(x)$ have $d + 1$ coefficients that are 1.

Alice then needed to compute $v \equiv st^{-1} \bmod p$ in order to get something large to use in her public key. Again, we can do this in R_p, calculating:

$$\left[\overline{v(x)}\right] = \left[\overline{s(x)}\right]\left[\overline{t(x)}\right]^{-1} \in R_p.$$

The resulting polynomial $v(x)$ is "large", as it will have quite large random looking coefficients mod p.

Alice now has a private key $([s(x)], [t(x)])$ and a public key $\left(N, p, q, d, \left[\overline{v(x)}\right]\right)$.

Bob was able to send a message m to Alice by converting to a small integer noise value r and sending Alice $e \equiv av + r \bmod p$, for some random small-ish integer a. He is still able to do this

in our new setting, he instead converts his message into a "small-ish" $[r(x)] \in R$ with coefficients satisfying $-\frac{q}{2} < r_i < \frac{q}{2}$ and sends

$$\left[\overline{e(x)}\right] \equiv \left[\overline{qa(x)v(x) + r(x)}\right] \in R_p.$$

Alice decrypted by computing $b \equiv te \bmod p$ and then $r \equiv t^{-1}b \bmod s$ (where this inverse is mod s). Alice should be able to do this in our new setup by computing:

$$\left[\overline{b(x)}\right] = \left[\overline{t(x)e(x)}\right] = \left[\overline{qa(x)s(x) + t(x)r(x)}\right] \in R_p$$

and then computing

$$\left[\overline{r(x)}\right] = \left[\overline{t(x)}\right]^{-1} \left[\overline{b(x)}\right] \in R_q.$$

Taking care of the "smalls" and the "larges" gives the following scheme:

NTRU:

1. Alice chooses N, p, q, d with N, p, q prime, $p \neq q, N$ and $p > (6d + 1)q$.
2. Alice chooses polynomials $[s(x)], [t(x)] \in R$ such that:
   - Both classes are invertible in R_p and R_q.
   - The polynomial $s(x)$ has d coefficients that are 1, d coefficients that are -1 and the rest of the coefficients 0.
   - The polynomial $t(x)$ has $d + 1$ coefficients that are 1, d coefficients that are -1 and the rest of the coefficients 0.

   Her private key is the pair $([s(x)], [t(x)]) \in R^2$ and her public key is $\left(N, p, q, d, \left[\overline{v(x)}\right]\right)$, with

   $$\left[\overline{v(x)}\right] = \left[\overline{s(x)}\right] \left[\overline{t(x)}\right]^{-1} \in R_p.$$

3. Bob encrypts a message m by converting to $[r(x)] \in R$ with coefficients $-\frac{q}{2} < r_i < \frac{q}{2}$. He then sends $\left[\overline{e(x)}\right] = \left[\overline{qa(x)v(x) + r(x)}\right] \in R_p$, for some random choice of $a(x) \in R$ (also having d coefficients that are 1, d coefficients that are -1 and the rest 0).
4. Alice decrypts by first computing $\left[\overline{b(x)}\right] = \left[\overline{t(x)e(x)}\right] \in R_p$ and then by computing $\left[\overline{r(x)}\right] = \left[\overline{t(x)}\right]^{-1} \left[\overline{b(x)}\right] \in R_q$.

We won't prove that the decryption algorithm works in this course, since it is a similar calculation to the one we did for the "simple noisy scheme" (but more tedious). The key point is that in the previous scheme, the private key parameters were chosen to be small enough so that Alice found the equality of integers $b = as + tr$ (which then gives the correct value mod s necessary to retrieve r). In NTRU, the parameters have once again been chosen so that we get an equality over the integer structure:

$$[b(x)] = [a(x)s(x) + t(x)r(x)] \in R,$$

which lets Alice recover the correct polynomial $[r(x)]$ when reducing mod q.

Example 7.21. *Alice chooses parameters* $(N, p, q, d) = (7, 41, 3, 2)$. *Note that* N, p, q *are prime,* $p \neq q, N$ *and* $p > (6d+1)q$ *(so this is a valid choice).*

She chooses her private key to be $([s(x)], [t(x)]) \in R$, *where:*

$$s(x) = x^6 + x^4 - x^2 - x \qquad t(x) = x^6 - x^4 + x^3 + x^2 - 1.$$

(Note that these have the correct numbers of coefficients that are ± 1*).*

Her public key is then $\left(N, p, q, d, \left[\overline{v(x)}\right]\right) = \left(7, 41, 3, 2, \left[\overline{20x^6 + 40x^5 + 2x^4 + 38x^3 + 8x^2 + 26x + 30}\right]\right)$, *since:*

$$\left[\overline{v(x)}\right] = \left[\overline{s(x)}\right]\left[\overline{t(x)}\right]^{-1} = \left[\overline{20x^6 + 40x^5 + 2x^4 + 38x^3 + 8x^2 + 26x + 30}\right] \in R_p.$$

Bob wishes to send a message m *which he has converted to:*

$$[r(x)] = [-x^5 + x^3 + x^2 - x + 1] \in R.$$

He chooses the random element

$$[a(x)] = [x^6 - x^5 + x - 1] \in R$$

and sends

$$\left[\overline{e(x)}\right] = \left[\overline{qa(x)v(x) + r(x)}\right] = \left[\overline{31x^6 + 19x^5 + 4x^4 + 2x^3 + 40x^2 + 3x + 25}\right] \in R_p.$$

Alice decrypts by computing

$$\left[\overline{b(x)}\right] = \left[\overline{t(x)e(x)}\right] = \left[\overline{x^6 + 10x^5 - 8x^4 - x^3 - x^2 + x - 1}\right] \in R_p.$$

Then

$$\left[\overline{r(x)}\right] = \left[\overline{-x^5 + x^3 + x^2 - x + 1}\right] \in R_q,$$

giving $r(x) = -x^5 + x^3 + x^2 - x + 1$ *as expected.*

So why is NTRU believed to be secure? Well just as before, it's enough for Eve to find two "small" elements $\left[\overline{S(x)}\right], \left[\overline{T(x)}\right] \in R_p$ such that $\left[\overline{v(x)}\right] = \left[\overline{S(x)}\right]\left[\overline{T(x)}\right]^{-1} \in R_p$. Equivalently, she needs to solve $\left[\overline{T(x)v(x)}\right] = \left[\overline{S(x)}\right] \in R_p$, so that $[T(x)v(x)] = [S(x) + pR(x)] \in R$, for some $[R(x)] \in R$.

It then follows that the vector $\mathbf{X}(x) = ([S(x)], [T(x)]) \in R^2$ satisfies:

$$\mathbf{X}(x) = ([S(x)], [T(x)]) = ([T(x)v(x) - pR(x)], [T(x)]) = [T(x)]([v(x)], [1]) - [R(x)]([p], [0])$$

if we allow ourselves to treat elements of R as scalars (i.e. we consider R^2 as an R-**module**).

In other words, we find that $([S(x)], [T(x)])$ is a "short vector" in the "lattice" that is the R-span of the vectors $\mathbf{v}_1 = ([v(x)], [1])$ and $\mathbf{v}_2 = ([p], [0])$.

If we write everything in integer coordinates, using the $\mathbb{Z}$-basis $[1], [x], [x^2], ..., [x^{N-1}]$ for R, then we would find that the vector

$$\mathbf{X}(x) = ([S(x)], [T(x)]) \longmapsto (s_0, s_1, s_2, ..., s_{N-1}, t_0, t_1, t_2, ..., t_{N-1}) \in \mathbb{Z}^{2N}$$

is a short vector in the lattice $\mathcal{L}$ with basis matrix:

$$\left(\begin{array}{ccccc|ccccc} v_0 & v_1 & v_2 & ... & v_{N-1} & 1 & 0 & 0 & ... & 0 \\ v_{N-1} & v_0 & v_1 & ... & v_{N-2} & 0 & 1 & 0 & ... & 0 \\ v_{N-2} & v_{N-1} & v_0 & ... & v_{N-3} & 0 & 0 & 1 & ... & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ v_1 & v_2 & v_3 & ... & v_0 & 0 & 0 & 0 & ... & 1 \\ \hline p & 0 & 0 & ... & 0 & 0 & 0 & 0 & ... & 0 \\ 0 & p & 0 & ... & 0 & 0 & 0 & 0 & ... & 0 \\ 0 & 0 & p & ... & 0 & 0 & 0 & 0 & ... & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & ... & p & 0 & 0 & 0 & ... & 0 \end{array}\right).$$

This is a lattice of dimension $2N$, and since N is large we currently have no hope of attacking the Short Vector Problem for this lattice. Even the best known methods for reducing a lattice basis, e.g. the **LLL algorithm**, fail to give us the short vector we seek (it can only give us an approximate short vector, which in this case happens to be one of the vectors $(0, 0, ..., 0, p, 0, ..., 0)$ that is too long to be the shortest one).

And that's the end! Throughout this course we've seen a wide variety of encryption/signature schemes, some historical and some modern (literally still in use). This list was by no means complete...there are way more schemes to read about, and further topics in Cryptography. Hopefully you found the journey interesting!

8 Appendix

8.1 Relative frequency table for the English language

The following table (taken from `https://en.wikipedia.org/wiki/Letter_frequency`) gives relative frequencies for appearance of letters in the English language (from most frequent to least).

Letter	Relative Frequency (%)
E	12.7
T	9.1
A	8.2
O	7.5
I	7.0
N	6.7
S	6.3
H	6.1
R	6.0
D	4.3
L	4.0
U	2.8
C	2.8
M	2.4
W	2.4
F	2.2
G	2.0
Y	2.0
P	1.9
B	1.5
V	0.98
K	0.77
J	0.15
X	0.15
Q	0.095
Z	0.074

Most common digraphs: `TH HE AN RE ER IN ON AT ND ST ES EN OF TE ED OR TI HI AS TO`

Most common double letters: `LL EE SS OO TT FF RR NN PP CC`

8.2 Mutual Index of Coincidence table for Example 1.31

i	j	0	1	2	3	4	5	6	7	8	9	10	11	12
1	2	0.025	0.034	0.045	0.049	0.025	0.032	0.037	0.042	0.049	0.031	0.032	0.037	0.043
1	3	0.023	0.067	0.055	0.022	0.034	0.049	0.036	0.040	0.040	0.046	0.025	0.031	0.046
1	4	0.032	0.041	0.027	0.040	0.045	0.037	0.045	0.028	0.049	0.042	0.042	0.030	0.039
1	5	0.043	0.021	0.031	0.052	0.027	0.049	0.037	0.050	0.033	0.033	0.035	0.044	0.030
1	6	0.037	0.036	0.030	0.037	0.037	0.055	0.046	0.038	0.035	0.031	0.032	0.037	0.032
1	7	0.054	0.063	0.034	0.030	0.034	0.040	0.035	0.032	0.042	0.025	0.019	0.061	0.054
2	3	0.041	0.029	0.036	0.041	0.045	0.038	0.060	0.031	0.020	0.045	0.056	0.029	0.030
2	4	0.028	0.043	0.042	0.032	0.032	0.047	0.035	0.048	0.037	0.040	0.028	0.051	0.037
2	5	0.047	0.037	0.032	0.044	0.059	0.029	0.017	0.044	0.060	0.034	0.037	0.046	0.039
2	6	0.033	0.035	0.052	0.040	0.032	0.031	0.031	0.029	0.055	0.052	0.043	0.028	0.023
2	7	0.038	0.037	0.035	0.046	0.046	0.054	0.037	0.018	0.029	0.052	0.041	0.026	0.037
3	4	0.029	0.039	0.033	0.048	0.044	0.043	0.030	0.051	0.033	0.034	0.034	0.040	0.038
3	5	0.021	0.041	0.041	0.037	0.051	0.035	0.036	0.038	0.025	0.043	0.034	0.039	0.036
3	6	0.037	0.034	0.042	0.034	0.051	0.029	0.027	0.041	0.034	0.040	0.037	0.046	0.036
3	7	0.046	0.023	0.028	0.040	0.031	0.040	0.045	0.039	0.020	0.030	0.069	0.042	0.037
4	5	0.041	0.033	0.041	0.038	0.036	0.031	0.056	0.032	0.026	0.034	0.049	0.029	0.054
4	6	0.035	0.037	0.032	0.039	0.041	0.033	0.032	0.039	0.042	0.031	0.049	0.039	0.058
4	7	0.031	0.032	0.046	0.038	0.039	0.042	0.033	0.056	0.046	0.027	0.027	0.036	0.036
5	6	0.048	0.036	0.026	0.031	0.033	0.039	0.037	0.027	0.037	0.045	0.032	0.040	0.041
5	7	0.030	0.051	0.043	0.031	0.034	0.041	0.048	0.032	0.053	0.037	0.024	0.029	0.045
6	7	0.032	0.033	0.030	0.038	0.032	0.035	0.047	0.050	0.049	0.033	0.057	0.050	0.021

i	j	13	14	15	16	17	18	19	20	21	22	23	24	25
1	2	0.034	0.052	0.037	0.030	0.037	0.054	0.021	0.018	0.052	0.052	0.043	0.042	0.046
1	3	0.031	0.037	0.038	0.050	0.039	0.040	0.026	0.037	0.044	0.043	0.023	0.045	0.032
1	4	0.039	0.040	0.032	0.041	0.028	0.019	0.071	0.038	0.040	0.034	0.045	0.026	0.052
1	5	0.042	0.032	0.038	0.037	0.032	0.045	0.045	0.033	0.041	0.043	0.035	0.028	0.063
1	6	0.040	0.030	0.028	0.071	0.051	0.033	0.036	0.047	0.029	0.037	0.046	0.041	0.027
1	7	0.040	0.032	0.049	0.037	0.035	0.035	0.039	0.023	0.043	0.035	0.041	0.042	0.027
2	3	0.054	0.040	0.028	0.031	0.039	0.033	0.052	0.046	0.037	0.026	0.028	0.036	0.048
2	4	0.047	0.034	0.027	0.038	0.047	0.042	0.026	0.038	0.029	0.046	0.040	0.061	0.025
2	5	0.034	0.026	0.035	0.038	0.048	0.035	0.033	0.032	0.040	0.041	0.045	0.033	0.036
2	6	0.033	0.034	0.036	0.036	0.048	0.040	0.041	0.049	0.058	0.028	0.021	0.043	0.049
2	7	0.042	0.037	0.041	0.059	0.031	0.027	0.043	0.046	0.028	0.021	0.044	0.048	0.040
3	4	0.037	0.045	0.033	0.028	0.029	0.073	0.026	0.040	0.040	0.026	0.043	0.042	0.043
3	5	0.035	0.029	0.036	0.044	0.055	0.034	0.033	0.046	0.041	0.024	0.041	0.067	0.037
3	6	0.023	0.043	0.074	0.047	0.033	0.043	0.030	0.026	0.042	0.045	0.032	0.035	0.040
3	7	0.035	0.035	0.035	0.028	0.048	0.033	0.035	0.041	0.038	0.052	0.038	0.029	0.062
4	5	0.032	0.041	0.036	0.032	0.046	0.035	0.039	0.042	0.038	0.034	0.043	0.036	0.048
4	6	0.034	0.034	0.036	0.029	0.043	0.037	0.039	0.036	0.039	0.033	0.066	0.037	0.028
4	7	0.043	0.032	0.039	0.034	0.029	0.071	0.037	0.039	0.030	0.044	0.037	0.030	0.041
5	6	0.052	0.035	0.019	0.036	0.063	0.045	0.030	0.039	0.049	0.029	0.036	0.052	0.041
5	7	0.040	0.031	0.034	0.052	0.026	0.034	0.051	0.044	0.041	0.039	0.034	0.046	0.029
6	7	0.029	0.035	0.039	0.032	0.028	0.039	0.026	0.036	0.069	0.052	0.035	0.034	0.038

8.3 ASCII table

The following is a table of ASCII encodings for the standard alphabet (larger tables exist for other symbols in common use).

ASCII	Letter
01000001	A
01000010	B
01000011	C
01000100	D
01000101	E
01000110	F
01000111	G
01001000	H
01001001	I
01001010	J
01001011	K
01001100	L
01001101	M
01001110	N
01001111	O
01010000	P
01010001	Q
01010010	R
01010011	S
01010100	T
01010101	U
01010110	V
01010111	W
01011000	X
01011001	Y
01011010	Z