Topics in Discrete Mathematics: Introduction to Mathematical Cryptography.

Dan Fretwell

Spring semester 2021/22

1 Introduction: Classical encryption methods

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!

Cryptography is the "art of secret writing". In modern language, the main idea is that one can **encrypt** messages taken from a finite set M by applying an **encryption map**, $e : M \hookrightarrow C$ (with C being another finite 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 a message $m \in M$ **purely from** $c = e(m) \in C$.

How would I be able to view the message? You will be clever and know enough about the map e to be able to **efficiently** compute the corresponding **decryption map** $d : \mathrm{Im}(e) \to M$ satisfying $d \circ e = \mathrm{id}_M$. Then you can **decrypt** by computing $d(c) = (d \circ e)(m) = \mathrm{id}_M(m) = m$ and retrieve the original message.

How do I share information with selected people without compromising security? Good question. Most classical methods of encryption depend on sharing a **key**, a piece of information that allows construction of both of the maps e and d. For such methods the security depends heavily on keeping the shared key a secret.

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 2021 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 uses classical techniques to **encrypt** the message.

What would Caesar have done?

Let $M = C = \{\mathtt{A}, \mathtt{B}, \mathtt{C}, ..., \mathtt{Z}\}$. Choose $0 \leq k \leq 25$ and use encoding map e_k, the map that **shifts** a letter k places in the alphabet (cycling round from `Z` to `A` if necessary). Then k is the key and the decoding map is $d_k = e_{26-k}$ (i.e. shift forward by $26 - k$, alternatively shift backwards by k).

Taking $k = 3$, our example encrypts to:

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 relabel letters $\mathtt{A}, \mathtt{B}, ..., \mathtt{Z}$ as the numbers $0, 1, 2, ..., 25$ respectively (which we will do from now on) then M and C are both in bijection with $\mathbb{Z}/26\mathbb{Z}$ and e_k becomes the map $e_k(m) \equiv m + k \bmod 26$.

This method of encryption sufficed in Caesar's day, but it is primitive now. An attacker can try all 26 possibilities in no time and look for the decryption that makes sense (although for short messages there could be more than one possibility, see Exercise Sheet 1). Never-the-less, this is considered to be one of the first true methods of encryption, that set the path to better methods.

What would Mary Queen of Scots have done?

The Caesar shift can be generalised in many ways. We'll see a few well known ways here (see Exercise Sheet 1 for more). One way in particular is to realise that the map e_k defines a **permutation** of $\mathbb{Z}/26\mathbb{Z}$. What if we just apply an arbitrary permutation, i.e. choose $\sigma \in S_{26}$ and define $e_\sigma(m) = \sigma(m)$?

For example the permutation $\sigma = (0\ 1)(2\ 3)(4\ 5)...(24\ 25)$ produces the **substitution key**:

ABCDEFGHIJKLMNOPQRSTUVWXYZ
BADCFEHGJILKNMPORQTSVUXWZY

so that our example encrypts to:

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?

Mary Queen of Scots used such a cipher to conspire to assassinate Queen Elizabeth I in 1586 (the Babington Plot). Such ciphers are also popular in films/games/novels. For example the Sherlock Holmes book "The Adventure of the Dancing Men" featured the following **substitution key**:

Although there are many possible keys ($26! \approx 4 \times 10^{26}$) such a cipher is still quite weak by today's standards. For a large enough ciphertext statistical tools such as **frequency analysis** can be used to recover the message (i.e. that certain letters in the English language appear more frequently than others). Common words can also be guessed, for example `THE`, `AND` (and in this case `LIKE`).

What would Vigenere have done?

Another way to generalise the Caesar shift is to realise that we don't have to use the same shift throughout the message. We can instead make a string $\mathbf{k} = (k_0, ..., k_{n-1})$ of shifts and cycle through them. Mathematically, we encode a message $\mathbf{m} = (m_0, m_1, ..., m_l)$ via $e_{\mathbf{k}}(m_i) \equiv m_i + k_{i \bmod n} \bmod 26$.

For example the key $\mathbf{k} = (1, 2)$ encrypts our message as follows:

```
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 a generic key $\mathbf{k}$ is just a random list of shifts. However, we can create keys using **keywords** by changing to numbers as before. For example the keyword `BRISTOL` gives the key $\mathbf{k} = (1, 17, 8, 18, 19, 14, 11)$.

One good thing about the Vigenere cipher is that the number of keys varies with the length of the message, since there are 26^n keys of length n. However, there still exist statistical tools that can attack these ciphers for small enough key size and large enough message (e.g. the **Kasiski exam** and the **Index of Coincidence** use statistics to predict the key size).

What would the President of the USA do?

All methods so far have had weaknesses, although we haven't said much about them. It's natural to ask whether there is a provably secure encryption method.

The **one time pad** encrypts a message of length l using the Vigenere method, but uses a key $\mathbf{k}$ of length $n = l$ chosen uniformly at random.

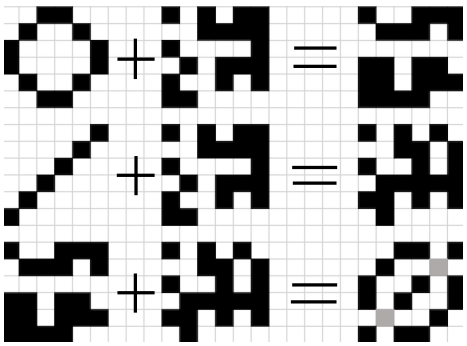
Why is this provably secure? Well, if we are allowing any letter to shift to any other letter in our message then **any** message can feasibly encrypt/decrypt to any other message of the same size. For example any three letter message is equally likely to decrypt to `DOG` as it is to `CAT`.
The first line of Alice's message could decrypt to:

```
EVE, I THINK YOUR TREATMENT OF YOUR FRIENDS ISV BAD.
```

Historically, the President of the USA used the One Time Pad to encrypt messages. Although provably secure, there are drawbacks:

- Alice must generate a **truly random** key $\mathbf{k}$. This is harder than it sounds!
- Alice must find a **secure** way to share the key with Bob. This is also harder than it sounds (the key is huge).
- Alice must only use the key **once**. Using the same key twice is bad, since if $e_{\mathbf{k}}(m_i) \equiv m_i + k_i \bmod 26$ and $e_{\mathbf{k}}(m_i') \equiv m_i' + k_i \bmod 26$ then subtracting gives $e_{\mathbf{k}}(m_i - m_i') \equiv m_i - m_i' \bmod 26$. This is independent of the key!

We can demonstrate this last point by picture, working mod 2. Adding the same random bit string to two pictures gives:



Adding the outputs gives the overlap of the two original pictures (grey cells are only shaded to emphasise this).

There are hundreds of classical encryption methods, this is just a small sample, e.g. you could look at what the Germans would have done (Enigma machine) or what the Zodiac Killer would have done (Homophonic encryption). One thing is clear...each classical method can be explained **mathematically**.

In other words, this is just the **tip** of a very large iceberg...

2 Diffie-Hellman: Asymmetric sharing

In the previous section we saw a couple of classical ways of encrypting data. However, in each case the security was heavily dependent on keeping the shared key secret, since this allowed efficient computation of both e and d. Such methods of encryption are called **symmetric**.

In the modern world we still use symmetric encryption methods (e.g. AES, Twofish and Serpent) since they are very secure and relatively efficient to implement. Most of the symmetric methods currently in use are sophisticated generalisations of the historical methods, some of which we saw in the previous section. However, we still have the issue of having to communicate the key with intended recipients.

Question: Can Alice and Bob agree on a shared key **without** either one sending it directly to the other?

Perhaps surprisingly, the answer is yes! First, we'll have to recap some things.

Definition 2.1. A **field** is a commutative ring F such that every $a \in F\backslash\{0\}$ is invertible, i.e. there exists $b \in F\backslash\{0\}$ such that $ab = ba = 1$.

There are many examples of fields, e.g. $\mathbb{Q}, \mathbb{R}, \mathbb{C}, \mathbb{C}(x), \mathbb{Q}(\sqrt{2})$. However, the fields that will interest us the most in this course are the **finite fields**, in particular $\mathbb{F}_p = \mathbb{Z}/p\mathbb{Z}$ for any prime p.

Theorem 2.2. *Let F be a field.*

- *The set $F^\times = F\backslash\{0\}$ is an abelian group under multiplication.*
- *If F is a finite field then $F^\times$ is a cyclic group of size $|F| - 1$.*

Proof. Left as an exercise (alternatively check Algebra 2 notes). □

Definition 2.3. A **primitive root** for a finite field F is any $g \in F$ that generates the group $F^\times$.

Example 2.4. $g = 2$ is a primitive root of $\mathbb{F}_5$, since $g^2 = 4$, $g^3 = 3$ and $g^4 = 1$ (so that g has order 4). It is not a primitive root of $\mathbb{F}_7$, since $g^3 = 1$. However $g = 3$ is a primitive root of $\mathbb{F}_7$, since $g^2 = 2$, $g^3 = 6$, $g^4 = 4$, $g^5 = 5$, $g^6 = 1$.

It is usually easy to find a primitive root of $\mathbb{F}_p$, even for large p. Quite often $g = 2, 3, 5$ works. Will they work for **infinitely** many p? This is the subject of a famous open conjecture.

Conjecture 2.5. *(Artin's primitive root conjecture) Let $g \in \mathbb{Z}$ not be -1 or a perfect square. Then g is a primitive root of $\mathbb{F}_p$ for infinitely many p.*

This conjecture is not known for even a single value of G, although it is known to follow from an even bigger and more famous open conjecture, the **Generalised Riemann Hypothesis**. Heuristics suggest that the density of primes such that $g = 2$ is a primitive root is $\prod_{p \text{ prime}} \left(1 - \frac{1}{p(p-1)}\right) = 0.373955....$

We are now ready to answer our question. Diffie-Hellman key exchange allows two people to produce a shared key in $\mathbb{F}_p^\times$ in a secure way. This can then be used as a key for a symmetric encryption scheme (although we will not see how in this course).

Diffie-Hellman key exchange

1. Alice and Bob agree on a prime p and a choice of primitive root g for $\mathbb{F}_p^\times$.
2. Alice chooses a random number $1 \leq k_A \leq p-1$ and Bob chooses a random number $1 \leq k_B \leq p-1$.
3. Alice sends $a = g^{k_A}$ to Bob and Bob sends $b = g^{k_B}$ to Alice.
4. Alice computes $s_A = b^{k_A}$ and Bob computes $s_B = a^{k_B}$ and both values are the shared key.

Notice that, although Alice and Bob have their own secret values k_A and k_B, neither needs to share their secret value with the other in order to carry out the procedure (we call such a method **asymmetric**). Also, it is not necessary to keep the values p and g secret.

Lemma 2.6. *Diffie-Hellman key exchange produces a shared key.*

Proof. Alice computes $s_A = b^{k_A} = (g^{k_B})^{k_A} = g^{k_B k_A} = g^{k_A k_B} = (g^{k_A})^{k_B} = a^{k_B}$, which is Bob's computed value s_B. □

Warning - While Diffie-Hellman works for all choices of (p, g, k_A, k_B) there are some bad choices (see Exercise Sheet 1).

Example 2.7. Alice and Bob agree to use $p = 37$ and $g = 2$ with secret values $k_A = 7$ and $k_B = 12$ respectively.

Alice sends the value $a = 17$ to Bob, since:

$$g^{k_A} = 2^7 \equiv 2^5 \times 2^2 \equiv (-5) \times 4 \equiv \mathbf{17} \bmod 37.$$

Bob sends the value $b = 26$ to Alice, since:

$$g^{k_B} = 2^{12} \equiv (2^5)^2 \times 2^2 \equiv (-5)^2 \times 4 \equiv \mathbf{26} \bmod 37.$$

Both then compute the shared value $s = 26$, since:

$$s_A = b^{k_A} = 26^7 = -11^7 \equiv -10^3 \times 11 \equiv 11 \times 10 \times 11 \equiv -11 \equiv \mathbf{26} \bmod 37$$
$$s_B = a^{k_B} = 17^{12} = 20^{12} \equiv 7^6 \equiv 12^3 \equiv (-4) \times 12 \equiv -48 \equiv \mathbf{26} \bmod 37$$

The above example might seem like a lot of work to do by hand, but computers are very quick at computing powers mod p (even for huge primes). Algorithms exist that can compute g^k in $O(\log\, k)$ time (i.e. the number of bits in the binary expansion of k roughly determines the overall run time). However, the **inverse** problem is very hard...

Definition 2.8. Let $a \in \mathbb{F}_p^\times$ for a fixed prime p. A discrete log for a with respect to primitive root g is $b \in \mathbb{Z}$ such that $a = g^b$.

We have to resist the temptation to write $b = \log_g(a)$ for now, since this is not well defined. A given $a \in \mathbb{F}_p^\times$ has **infinitely many** discrete logs (as with logs over $\mathbb{C}$).

Example 2.9. If $p = 11, g = 2$ and $a = 6$ then $b = 9$ is a discrete log for a, since:

$$g^b = 2^9 \equiv -16 \equiv \mathbf{6} \bmod 11.$$

However, $b = 19$ is also a discrete log for a, since:

$$g^b = 2^{19} \equiv 2^{10} \times 2^9 \equiv 2^9 \equiv \mathbf{6} \bmod 11.$$

In fact all numbers of the form $9 + 10m$ are discrete logs for a.

Theorem 2.10. *Let $a \in \mathbb{F}_p^\times$. Then the set of discrete logs for a with respect to g is a congruence class mod $(p-1)$. Further, there is a well defined group isomorphism*

$$\log_g : \mathbb{F}_p^\times \longrightarrow \mathbb{Z}/(p-1)\mathbb{Z}.$$

Proof. If b is a discrete log for a with respect to g then so is $b + (p-1)k$ for any $k \in \mathbb{Z}$, since by Fermat's little theorem:

$$g^{b+(p-1)k} = g^b g^{(p-1)k} = g^b (g^{p-1})^k = g^b = a.$$

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

We now know that there is a well defined map:

$$\log_g(a) : \mathbb{F}_p^\times \longrightarrow \mathbb{Z}/(p-1)\mathbb{Z}.$$

It is left as an exercise to prove that this is in fact a group homomorphism (i.e. to show that $\log_g(a_1 a_2) = \log_g(a_1) + \log_g(a_2)$). The fact that it is then a group isomorphism is clear, since $\log_g(g) = 1 + (p-1)\mathbb{Z}$ generates the RHS.

□

We already knew that $\mathbb{F}_p^\times \cong \mathbb{Z}/(p-1)\mathbb{Z}$, but a discrete log map gives one possible way of **explicitly** realising this isomorphism.

Returning to the Diffie-Hellman scheme, we might ask why it is secure? Put yourself in Eve's shoes. She only knows p and g and can only ever see g^{k_A} and g^{k_B}. To get the shared secret she has to construct $g^{k_A k_B}$ from this information.

The Diffie-Hellman Problem
Given a prime p, a primitive root g of $\mathbb{F}_p^\times$ and values g^m and g^n, compute g^{mn}.

This problem is generally thought to be hard to solve, the best known algorithms for solving it run in **exponential time** (so are extremely slow). Finding a **polynomial time** algorithm for this would be amazing, proving one doesn't exist would solve a Millennium problem (P vs NP) and earn you a million dollars. Either leads to eternal fame!

Warning - Practically speaking, the Diffie-Hellman Problem is only considered hard for **large** enough p (depending on the current level of technology) and generic m, n. For example, if p is a two digit prime then it's feasible to find m and n by hand, by simply computing enough powers of g. More sophisticated attacks exist for larger p. Typically, p should have around 2048 bits to be considered secure by current standards (i.e. around 617 digits).

You might wonder why we can't just compute m and n by taking discrete logs, since $\log_g(g^m) = m + (p-1)\mathbb{Z}$ and $\log_g(g^n) = n + (p-1)\mathbb{Z}$. This is because the following problem is also believed to be hard to solve.

The Discrete Log Problem
Given a prime p, a primitive root g of $\mathbb{F}_p^\times$ and $a \in \mathbb{F}_p^\times$, find $\log_g(a)$.

As expected, any algorithm that solves this problem quickly will also solve the Diffie-Hellman Problem quickly. Sadly, no such fast algorithm is known for large p. It is widely believed that both problems are **equivalently hard** to solve, but only partial results have been obtained in this direction.

Diffie-Hellman key exchange was introduced in 1976. The idea that security could depend on the **toughness** of a mathematical problem was revolutionary and would soon lead to new and exciting cryptosystems, as we will see.

In other words, the fact that Mathematics is **hard** can actually be a good thing...

3 RSA: The birth of public key cryptography

In the previous section we saw that Alice and Bob can agree on a shared secret key by using Diffie-Hellman key exchange. The security of this asymmetric scheme rests entirely on the fact that the Diffie-Hellman Problem is a tough mathematical problem.

Question: Is it possible for Alice and Bob to communicate securely **without** having to use a shared key?

For a long time it was believed that the only way to communicate securely was via symmetric schemes (and that the security depended on keeping the shared key secret). However, in 1977, Ron Rivest, Adi Shamir and Leonard Adleman discovered a new scheme that allowed efficient communication in an **asymmetric** way. The resulting scheme, RSA, provided a huge breakthrough in cryptography and is still one of the most widely used schemes.

Once again, the security will rely on the toughness of a mathematical problem. First, we recall a few facts.

For $N \geq 1$ we consider the commutative ring $\mathbb{Z}/N\mathbb{Z}$ under addition and multiplication. We're interested in the unit group $(\mathbb{Z}/N\mathbb{Z})^\times$.

Lemma 3.1. *For $N \geq 1$ the unit group $(\mathbb{Z}/N\mathbb{Z})^\times$ consists of classes generated by a that are coprime with N. Further, this group has size $\phi(N)$ with:*

$$\phi(N) = |\{1 \leq a \leq N \,|\, \gcd(a, N) = 1\}| = N \prod_{\text{primes } p|N} \left(1 - \frac{1}{p}\right).$$

Proof. The class of $a \in \mathbb{Z}$ is invertible mod N if and only if there exists $s, t \in \mathbb{Z}$ such that $as + Nt = 1$. If $\gcd(a, N) \neq 1$ then there is no solution whereas if $\gcd(a, N) = 1$ then Euclid's algorithm can be used to construct a solution. The first equation then follows since every class mod N has a unique representative in the range $1 \leq m \leq N$.

The second equation follows by Inclusion-Exclusion (see Exercise Sheet 1). □

Example 3.2. For $N = 14$ we have $(\mathbb{Z}/14\mathbb{Z})^\times = \{1, 3, 5, 9, 11, 13\}$, thus $\phi(14) = 6 = 14\left(1 - \frac{1}{2}\right)\left(1 - \frac{1}{7}\right)$. One can check that $3^{-1} = 5, 9^{-1} = 11$ and $13^{-1} = 13$.

For $N = p$ prime we have $(\mathbb{Z}/p\mathbb{Z})^\times = \mathbb{F}_p \backslash \{0\} = \mathbb{F}_p^\times$, as we saw before. This agrees with $\phi(p) = p\left(1 - \frac{1}{p}\right) = p - 1$.

Similarly to Diffie-Hellman, we are going to use exponentiation maps mod N to encode. But now we want to be able to decode too, and so we'll need to know when we can **invert** these maps.

Proposition 3.3. *For $N \geq 1$ and $e \geq 1$ the map:*

$$\begin{aligned} f_e : (\mathbb{Z}/N\mathbb{Z})^\times &\longrightarrow (\mathbb{Z}/N\mathbb{Z})^\times \\ m &\longmapsto m^e \end{aligned}$$

is invertible if and only if e is coprime with $\phi(N)$.

Proof. If e is coprime with $\phi(N)$ then there exist $d, t \in \mathbb{Z}$ such that $ed+\phi(N)t = 1$ (using Euclid's algorithm). We claim that the map f_d is the inverse of f_e.

Note that since $f_d \circ f_e = f_e \circ f_d$ we only have to check that one of these is the identity map. The claim then follows since:

$$(f_d \circ f_e)(m) = f_d(m^e) = m^{ed} = m^{1-\phi(N)t} = m(m^{\phi(N)})^{-t} = m,$$

since $m^{\phi(N)} = 1$ for any $m \in (\mathbb{Z}/N\mathbb{Z})^\times$, by Lagrange's theorem (or by Euler's generalisation of Fermat's little theorem).

For each prime p dividing $\phi(N)$ there exists an element $a \in (\mathbb{Z}/N\mathbb{Z})^\times$ of order p (by Cauchy's theorem). If there exists such a prime that also divides e then $f_e(a) = a^e = (a^p)^{\frac{p}{e}} = 1$. However, $f_e(1) = 1$ too and so f_e is not invertible. □

Note that the maps we are looking at here are slightly different to the ones we used in Diffie-Hellman. There we fixed the base g and applied an arbitrary power. Here we are fixing the power and applying it to an arbitrary base.

Let's now think a little about how we might use these maps to encrypt securely.

Bad idea

Alice chooses $N = p$ and wishes to send a message $m \in (\mathbb{Z}/p\mathbb{Z})^\times$ to Bob. She chooses a random $1 \leq e \leq p-1$ and sends $c = f_e(m)$ to Bob. He can't retrieve the message since he doesn't know p or e, so he asks for those too. Alice sends them on, Bob computes $d = e^{-1} \bmod p-1$ and hence computes $f_d(c) = m$.

While Bob can compute everything necessary in polynomial time (using Euclid's algorithm), there is no security in this method at all! The problem is that Alice has also given Eve everything she needs to do the exact same computation Bob did.

We need to somehow ensure that Bob has enough information to invert e quickly, but that Eve **can't** do it feasibly given what she knows. The clever idea of RSA is to instead let Bob choose N, rather than Alice, to guarantee this.

How should Bob choose N? We can't use $N = p$ because of the issue above, everyone would be able to see it. However, a product of two primes works very well!

Good idea - The RSA scheme

1. Bob chooses two secret primes p, q. He computes the **semiprime** $N = pq$ and chooses e coprime with $\phi(N) = (p-1)(q-1)$. These are sent to Alice.
2. She encrypts a message $m \in (\mathbb{Z}/N\mathbb{Z})^\times$ by computing $c = f_e(m)$ (she knows e and N). This is sent to Bob, who can compute $d = e^{-1} \bmod (p-1)(q-1)$ and hence compute $f_d(c) = m$.

As with any scheme there are bad choices of parameters (see Exercise Sheet 1).

Example 3.4. Bob chooses $p = 7$ and $q = 13$, so that $N = 91$ and $\phi(N) = 6 \times 12 = 72$. He chooses $e = 5$ and releases the values (e, N) to Alice.

Alice wishes to send the message $m = 11$ to Bob. She sends the encrypted message:

$$c = f_e(m) = 11^5 \equiv (11^2)^2 \times 11 \equiv 30^2 \times 11 \equiv -110 \equiv \mathbf{72} \bmod 91.$$

Bob receives $c = 72$ and computes

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

He then decrypts to get the original message:

$$\begin{aligned} m = f_d(c) = 72^{29} &\equiv -19^{29} \equiv -(-3)^{14} \times 19 \equiv -3^{14} \times 19 \equiv -(-10)^3 \times 9 \times 19 \\ &\equiv 1000 \times 9 \times 19 \equiv -9 \times 19 \equiv -171 \equiv \mathbf{11} \bmod 91. \end{aligned}$$

Why is the RSA scheme secure? Eve only knows N, e and c and has to get back to the message m.

The RSA Problem
Given $N = pq$ (for two secret primes p, q), e coprime to $\phi(N)$ and $c \equiv m^e \bmod N$ for some $m \in (\mathbb{Z}/N\mathbb{Z})^\times$, compute m (i.e. invert the map f_e).

As with the Diffie-Hellman Problem, this problem is also thought to be hard to solve. So why can Bob solve it easily? He knows the values of the primes p, q and so is able to form the number $\phi(N) = (p-1)(q-1)$. Knowing this, he can find the necessary inverse $d = e^{-1} \bmod (p-1)(q-1)$ and hence solve the RSA problem in **polynomial time**!

For Eve to replicate what Bob was able to do she would need to be able to compute the number $\phi(N) = (p-1)(q-1)$. The only known way to do this from the information she knows is to find p and q.

The Semiprime Factoring Problem
Given a semiprime $N = pq$, find the primes p and q (i.e. factor N).

It is now clear that if one can solve this problem efficiently (i.e. in polynomial time) then the RSA Problem would be easy to solve too. But how well can

we factor an **arbitrary** integer N? There's of course the ancient method of **trial division**, simply try to divide by every prime up to $\sqrt{N}$, but this is not an efficient algorithm for large N (it runs in **exponential time**). Throughout history we have developed many better algorithms for factoring, but the best ones we have (e.g. the Number Field Sieve) run in **sub-exponential** time and so are still too **slow** to attack RSA.

Warning - Practically speaking, the Semiprime Factoring Problem is only considered hard for **large enough** primes p and q (would you really have trouble factoring 15 or 91?). Typically, the public modulus N should have around 2048 bits to be considered secure by current standards (i.e. around 617 digits), although 1024 bits is also considered fine since the world record for semiprime factoring is 829 bits, (i.e. around 250 digits).

The RSA scheme was the first example of **public key cryptography**. In such a scheme everyone has two keys, **public** and **private**. Anyone can easily encrypt and communicate with someone using their public key, but only they can see their private key. For example, in RSA a public key is a pair (e, N) and the corresponding private key is the value $d = e^{-1} \bmod (p-1)(q-1)$.

Recall that for most classical systems the security depended on keeping the shared key secret. The security of a public key scheme depends on the fact that decryption without the private key is **infeasible**, but with the private key it is **fast**. Everyone theoretically knows the information needed to decrypt messages, but practically Mathematics makes it hard to get at!

Many public key schemes are constructed so that Eve has to solve a hard mathematical problem in order to decrypt, but Bob can turn it into an easy mathematical problem using his private key.

In other words, Mathematics is hard but Bob finds it **easy**...

4 ElGamal: A scheme for any finite group

When RSA was first released back in 1977 it was a revelation that Mathematics can be used to provide security. We can force the attacker, Eve, to have to solve a hard problem (e.g. the RSA Problem) to get at our secrets...whereas we can sit back and relax, since we have a built a secret "trapdoor" into the problem that makes it easy (e.g. we know the factors of N).

When we looked at the Diffie-Hellman key exchange, we saw that the security was based on the difficulty of the Diffie-Hellman problem, which is roughly related to the Discrete Log Problem.

Question: Is there a public key system whose security is based on the Discrete Log Problem?

The answer is yes, and in fact we'll be able to create one for any **finite cyclic group**!

First we need to know what discrete logs are in this setting. Let's go back to the case of $\mathbb{F}_p^\times$. A discrete log of $a \in \mathbb{F}_p^\times$ only made sense once we fixed a primitive root g, 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 4.1. Let G be a finite cyclic group. A **discrete log** of $h \in G$ with respect to generator g is $k \in \mathbb{Z}$ such that $h = g^k$.

Warning - Be careful, when we write g^k this means to apply the group operation k times to g. It doesn't necessarily mean that g is being multiplied by itself (e.g. the group operation might be addition).

Recall that in order to get a unique discrete log, we had to restrict to a class in $\mathbb{Z}/(p-1)\mathbb{Z}$ (see Theorem 2.10). Something very similar happens for cyclic groups.

Theorem 4.2. *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 is a congruence class mod $|G|$. Further, there is a well defined group isomorphism*

$$\log_g : G \longrightarrow \mathbb{Z}/|G|\mathbb{Z}.$$

Proof. If k is a discrete log for h with respect to g then so is $k + |G|m$ for any $m \in \mathbb{Z}$, since by Lagrange's theorem:

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

Now suppose that k' 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 $1 \leq k < k' \leq |G| - 1$. Then $h = g^k = g^{k'}$, implying that $g^{k'-k} = \mathrm{id}$. But this contradicts the fact that g is a primitive root, since then g has order dividing $k' - k < |G|$.

It is now clear that there is a well defined map:

$$\log_g : G \longrightarrow \mathbb{Z}/|G|\mathbb{Z}.$$

It is left as an exercise to prove that this is in fact a group homomorphism (i.e. that $\log_g(h_1h_2) = \log_g(h_1) + \log_g(h_2)$). The fact that it is then a group isomorphism is clear, since $\log_g(g) = 1 + |G|\mathbb{Z}$ generates the RHS. □

Example 4.3. Let $G = \mathbb{F}_p^\times$. Then the discrete log of $a \in G$ with respect to generator g is the same thing as the discrete log of a with respect to primitive root g. For example, if $p = 11$ and $g = 2$ then $\log_2(6) = 9 + 10\mathbb{Z}$ since $2^9 \equiv 2^{-1} \equiv \mathbf{6} \bmod 11$.

Let $G = \mathbb{Z}/10\mathbb{Z}$ under addition. Then $\log_7(3) = 9 + 10\mathbb{Z}$ since $7 \times 9 = 63 \equiv \mathbf{3} \bmod 10$.

Let $G = \mu_{10}$, the group of 10th roots of unity under multiplication. The discrete log of $h = e^{\frac{3\pi i}{10}}$ with respect to $\zeta = e^{\frac{7\pi i}{10}}$ is $\log_\zeta(h) = 9 + 10\mathbb{Z}$ since $\zeta^9 = e^{\frac{63\pi i}{10}} = e^{\frac{3\pi i}{10}} = h$.

You might not have realised this but all three of the above calculations were for the "same" group (cyclic of size 10), but the calculations feel completely different (at least the first one should feel very different to the other two). This shows that even though two cyclic groups may be **isomorphic**, discrete log calculations can be **completely different**.

This all might seem weird (at least to an algebraist)...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.

The ElGamal scheme

1. Bob chooses a finite cyclic group G of size q and fixes a generator g. He chooses a secret value $1 \leq k \leq |G|$ and computes $h = g^k$. Bob's public key is (G, q, g, h) and his private key is k.
2. Alice encrypts a message $m \in G$ by choosing a secret value $1 \leq s \leq |G|$ and computing $c_1 = g^s$ and $c_2 = mh^s$ (she knows this information). She sends $c = (c_1, c_2)$ to Bob.
3. Bob decrypts by computing $m = c_2c_1^{-k}$.

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

Lemma 4.4. *ElGamal decryption works.*

Proof. This is a simple calculation:

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

□

Example 4.5. Let's work with the multiplicative group $G = \mathbb{F}_{31}^{\times}$. Bob chooses generator $g = 3$ and private key $k = 11$. Then Bob's public key is $(\mathbb{F}_{31}^{\times}, 31, 3, 13)$, since:

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

Alice wishes to send the message $m = 7$. She chooses secret value $s = 4$ and computes that

$$c_1 = 3^4 \equiv \mathbf{19} \bmod 31$$
$$c_2 = 7 \times 13^4 \equiv 7 \times 14^2 \equiv 70 \equiv \mathbf{8} \bmod 31.$$

Then she sends $(19, 8)$ to Bob.

Bob decrypts by computing

$$\begin{aligned} m = 8 \times 19^{-11} \equiv 8 \times 19^{19} &\equiv 8 \times 19 \times 20^9 \equiv 8 \times 19 \times 4^9 \times 5^9 \\ &\equiv 8 \times 19 \times 2^3 \equiv 2 \times 19 \equiv \mathbf{7} \bmod 31. \end{aligned}$$

A few remarks:

- 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 ElGamal .
- We viewed messages as elements of $m \in G$, but we didn't explain why 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 ElGamal 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 $c_2 c_1^{-k}$. However, Eve does not know the value of k and so the only clear way for her to get this is to calculate $\log_g(h) = k + q\mathbb{Z}$. But this is an instance of the **Discrete Log Problem** for G.

The Discrete Log Problem (for cyclic groups):
Given a cyclic group G, a generator g and $a \in G$, find $\log_g(a)$.

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

Example 4.6. Let $G = \mathbb{Z}/N\mathbb{Z}$ (under addition) and choose g coprime with N. Then g generates G. Solving the Discrete Log Problem for this setup means being able to solve the congruence $gx \equiv a \bmod N$ (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 $g = 7$. Then $\log_7(9) = 5 + 13\mathbb{Z}$ since $7x \equiv 9 \bmod 13$ has solution $x \equiv 9 \times 7^{-1} \equiv 18 \equiv \mathbf{5} \bmod 13$. Surely enough $7 + 7 + 7 + 7 + 7 = 7 \times 5 = 35 \equiv \mathbf{9} \bmod 13$.

Given the above example it should be clear that you should **never** use $G = \mathbb{Z}/N\mathbb{Z}$ with the ElGamal scheme. The Discrete Log Problem is extremely simple to solve. In stark contrast, we saw that the Discrete Log Problem for $\mathbb{F}_p^\times$ is believed to be difficult, and so these groups are **good** choices for G.

In part two of this course you will see another family of good choices for G, coming from points on **elliptic curves** over finite fields. Much less is known about the Discrete Log Problem for these groups, due to the geometric nature of their group operation. Elliptic curves are a current favourite choice of group structure in cryptography, in particular they are currently used in a lot of **cryptocurrency** protocols (e.g. Bitcoin).

In other words, Mathematics is only hard if Bob makes **good choices** for G...

5 Digital Signatures: Proving that you are you

We've seen that public key cryptography provides a great way for people to communicate in an asymmetric way. But it can do way more than this.

One problem with symmetric schemes is that finding the shared key immediately allows Eve to read all messages. Another problem is that knowing the shared key also lets Eve pretend to be Alice or Bob, without immediate detection!

Most public key schemes often allow the sender to add a **digital signature**, providing **authentication** that the message was indeed sent by that person. How might this be possible?

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, **noone** except Alice should ever be given access to Alice's private key! (Otherwise all of her messages can be read).

Good idea

Alice realises that she somehow has to signal to Bob that the sender knows Alice's private key, without telling him or anyone else what it is.

She decides to do something clever. She chooses a second message m_0 and decrypts it using her private key (something that only Alice can feasibly do) to give m_1. She then sends Bob the original message m with the **signature** (m_0, m_1) (after encrypting using Bob's public key).

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 noone else can feasibly decrypt a message using only Alice's public key.

Let's look at two examples of digital signature schemes, based on the RSA and ElGamal schemes.

The RSA signature scheme

1. Alice chooses an RSA public/private key:
   - (e, N) consisting of a semiprime N and $1 \leq e \leq N$ coprime with N,
   - the secret value $d = e^{-1} \bmod (p-1)(q-1)$ (private key).
2. In order to sign a message she first chooses an auxiliary message $m_0 \in (\mathbb{Z}/N\mathbb{Z})^\times$ and decrypts using her private key to give $m_1 = f_d(m_0)$. When sending an encrypted message to Bob (using his public key), she signs it with the pair (m_0, m_1).
3. Bob decrypts the encrypted message (using his private key) and verifies the signature by checking that $f_e(m_1) = m_0$.

Example 5.1. Alice chooses RSA public key $(e_A, N_A) = (13, 143)$. Her private key is $d_A = 37$, since $\phi(N_A) = 120$ and $e_A d_A = 481 \equiv \mathbf{1} \bmod 120$.

Bob chooses RSA public key $(e_B, N_B) = (23, 55)$. His private key is $d_B = 7$, since $\phi(N_B) = 40$ and $e_B d_B = 161 \equiv \mathbf{1} \bmod 40$.

To encrypt the message $m = 2$ Alice uses Bob's public key:

$$c = f_{e_B}(m) = 2^{23} \equiv 9^4 \times 2^{-1} \equiv 26^2 \times 28 \equiv 13^2 \times 112 \equiv 4 \times 2 \equiv \mathbf{8} \bmod 55$$

To sign, she chooses auxiliary message $m_0 = 17$ and decrypts using her private key to give:

$$\begin{aligned} m_1 = f_{d_A}(m_0) = 17^{37} &\equiv 3^{18} \times 17 \equiv 14^3 \times 17 \equiv 7^3 \times (-7) \\ &\equiv -57 \times 7 \equiv -399 \equiv \mathbf{30} \bmod 143 \end{aligned}$$

Alice sends Bob the triple $(c, m_0, m_1) = (8, 17, 30)$.

Bob decrypts $c = 8$ using his private key to get the original message:

$$m = f_{d_B}(c) = 8^7 \equiv 9^3 \times 8 \equiv 81 \times 72 \equiv 26 \times 17 \equiv 442 \equiv \mathbf{2} \bmod 55.$$

He then computes, using Alice's public key:

$$\begin{aligned} m_1^{e_A} = 30^{13} &\equiv 3^{13} \times 10^{13} \equiv 3^{13} \times 10 \equiv 14^2 \times 3 \times 10 \\ &\equiv 53 \times 3 \times 10 \equiv 160 \equiv \mathbf{17} \bmod 143. \end{aligned}$$

The signature is then verified since $m_1^{e_A} = m_0 = 17$.

In practice, Alice can use the actual message m in order to sign. She clearly cannot take $m_0 = m$ (otherwise everyone would see the message!) but can instead apply a **hash function** H to m and take $m_0 = H(m)$. 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 ElGamal scheme, although these are a little more complicated than RSA signatures. The ones we will see **only** work for the cyclic groups $G = \mathbb{F}_p^\times$.

The ElGamal signature scheme

1. Alice chooses an ElGamal public/private key, i.e. a tuple $(\mathbb{F}_p^\times, g, h)$ with p prime, g a primitive root mod p and $h = g^k$ for some secret value $1 \leq k \leq p-1$ (Alice's private key).
2. In order to sign a message she chooses an auxiliary message $m_0 \in \mathbb{F}_p^\times$ and a secret value $1 \leq s \leq p-1$ coprime to $p-1$. She lets $m_1 = g^s$ and $m_2 \equiv (m_0 - km_1)s^{-1} \bmod p-1$. When sending an encrypted message to Bob (using his public key), she signs it with the triple (m_0, m_1, m_2).
3. Bob decrypts the encrypted message (using his private key) and verifies the signature by checking that $h^{m_1} m_1^{m_2} = g^{m_0}$.

Proposition 5.2. *The verification step is theoretically correct.*

Proof. This is because:

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

□

Let's see an example.

Example 5.3. Alice chooses ElGamal public key $(\mathbb{F}_{p_A}^\times, g_A, h_A) = (\mathbb{F}_{23}^\times, 5, 22)$ (her private key is $k_A = 11$).

Bob chooses ElGamal public key $(\mathbb{F}_{p_B}^\times, g_B, h_B) = (\mathbb{F}_{17}^\times, 3, 11)$ (his private key is $k_B = 7$).

To encrypt the message $m = 13$ Alice uses Bob's public key with secret value $s_1 = 3$:

$$c_1 = g_B^{s_1} = 3^3 \equiv 10 \bmod 17$$
$$c_2 = m h_B^{s_1} = 13 \times 11^3 \equiv 4 \times 6^3 \equiv 24 \times 2 \equiv \mathbf{14} \bmod 17$$

To sign she chooses auxiliary message $m_0 = 19$, secret value $s_2 = 9$ and com-

putes:

$$\begin{aligned} m_1 &= g_A^{s_2} = 5^9 \equiv 2^4 \times 5 \equiv 80 \equiv 11 \bmod 23 \\ m_2 &\equiv (m_0 - k_A m_1) s_2^{-1} \equiv (19 - 11 \times 11) \times 9^{-1} \equiv (-102) \times 5 \\ &\equiv -14 \times 5 \equiv 40 \equiv \mathbf{18} \bmod 22 \end{aligned}$$

Alice sends Bob the quintuple $(c_1, c_2, m_0, m_1, m_2) = (10, 14, 19, 11, 18)$.

Bob decrypts $(c_1, c_2) = (10, 14)$ using his private key to get the original message:

$$\begin{aligned} c_2 c_1^{-k_B} = 14 \times 10^{-7} \equiv (-3) \times 12^7 \equiv 3 \times 5^7 \equiv (-2) \times 5^6 \equiv -2 \times 8^3 \\ \equiv -16 \times 64 \equiv \mathbf{13} \bmod 17. \end{aligned}$$

He then computes, using Alice's public key:

$$\begin{aligned} h_A^{m_1} m_1^{m_2} &= 22^{11} \times 11^{18} \equiv -11^{18} \equiv -6^9 \equiv -13^4 \times 6 \equiv -8^2 \times 6 \\ &\equiv -64 \times 6 \equiv \mathbf{7} \bmod 23 \\ g_A^{m_0} &= 5^{19} \equiv 2^9 \times 5 \equiv 9 \times 16 \times 5 \equiv -16 \equiv \mathbf{7} \bmod 23 \end{aligned}$$

The two values match and so the signature is verified.

Why is this signature scheme secure? Eve only knows p and $h = g^k$, and in order to forge a signature she needs to be able to find $a, b, c \in \mathbb{Z}$ such that $h^b b^c = g^a$. She can then sign a message **fraudulently** by sending Bob the triple $(m_0, m_1, m_2) = (a, b, c)$.

We must justify that it is hard to solve this equation. Taking logs with respect to g gives:

$$b \log_g(h) + c \log_g(b) \equiv a \bmod p - 1.$$

The only known way to solve this equation is for Eve to pick values of a and b, compute $\log_g(h)$ and $\log_g(b)$, and then solve for c. Even if b is chosen well, Eve would still need to calculate $\log_g(h)$, which is a **generic** instance of the Discrete Log Problem (if Alice chose her keys well). Thus **only** Alice should feasibly be able to sign the messages that she sends.

There are of course **many** other signature schemes, some based on other public key schemes. In the second part of this course you will see how **elliptic curves** provide secure signature schemes.

In other words, mathematics can sign your name in **many** different ways....

6 Knapsack schemes: A postquantum precursor

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 ElGamal 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 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 and ElGamal 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.

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 6.1. 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 6.2. The sequence $a_n = 2^{n-1}$ is superincreasing since for any $1 \leq m \leq n$:

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

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 6.3. *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 $j_k < j_{h-1} < ... < j_1$. 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

$$s \leq a_{j_1} + ... + a_1 < a_{j_1 + 1} \leq a_{i_1} \leq a_{i_1} + ... + a_1.$$

Thus $a_{i_1} = a_{j_1}$ as expected. □

Example 6.4. 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 6.5. 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 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 $\mathbf{x} = (x_1, x_2, ..., x_n)$, 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 6.6. *The decryption algorithm works.*

Proof. Bob receives $c = \sum_{1 \leq t \leq n} x_t b_t$ and the claim is that $\mathbf{x}$ 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 $M > \sum_{1 \leq t \leq n} a_t$ we have equality $c' = \sum_{1 \leq t \leq n} x_t a_t$, proving our claim. □

Example 6.7. 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 $c' \equiv 6 * 80 \equiv \mathbf{25} \bmod 65$. 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.

Definition 6.8. Let $\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_n$ be a basis for $\mathbb{R}^n$. The **lattice** generated by this basis is the set $\mathcal{L} = \{\alpha_1\mathbf{v}_1 + \alpha_2\mathbf{v}_2 + ... + \alpha_n\mathbf{v}_n \,|\, \alpha_i \in \mathbb{Z}\}$.

Given a generic Knapsack Problem, for a sequence $b_1, b_2, ..., b_n$ and a target $s = \sum_{1\leq t\leq n} x_t b_t$, we can form the lattice $\mathcal{L} \subset \mathbb{R}^{n+1}$ generated by the rows $\mathbf{r}_1, \mathbf{r}_2, ..., \mathbf{r}_{n+1}$ of the matrix:

$$\begin{pmatrix} 2 & 0 & 0 & \dots & b_1 \\ 0 & 2 & 0 & \dots & b_2 \\ 0 & 0 & 2 & \dots & b_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & 1 & 1 & s \end{pmatrix}.$$

Since $s = \sum_{1\leq t\leq n} x_t b_t$ we note that the vector $\mathbf{v} = (\sum_{1\leq t\leq n} x_t\mathbf{r}_t) - \mathbf{r}_{n+1} = (2x_1 - 1, 2x_2 - 1, ..., 2x_n - 1, 0) \in \mathcal{L}$. Notice that since each $x_i \in \{0, 1\}$ this vector has Euclidean length $||\mathbf{v}|| = \sqrt{n}$. For reasons beyond this course, this is likely to be the **shortest** non-zero vector in $\mathcal{L}$.

The Shortest Vector Problem
Given a lattice $\mathcal{L}$, determine a vector $\mathbf{v} \in \mathcal{L}\backslash\{0\}$ that minimises $||\mathbf{v}||$.

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.

In other words, **postquantum cryptography** is on the rise...

7 Secret Sharing: The digital way

Often it is the case that someone wishes to share access to a secret with a group of people. Not only that but the secret might be valuable and so they might wish for it to be hard to access without enough people being present.

For example, Alice is head of a bank that has a huge vault full of money. She might want employees to have access to the vault, but for noone to have access by themselves. In order to guarantee this she could write down the combination and divide it into pieces, strategically distributing pieces among employees. This isn't great though, everyone sees a piece of the combination and so learns something.

Instead she could employee a unique key and have special locks installed that can only be unlocked using two different keys, say. This is much better, although the number of keys could get unwieldy.

Can we design such a system for sharing secrets **digitally**? Once again the answer is yes, and Mathematics comes to the rescue!

To be more precise we would like a way to take a digital secret S and to associate with it n pieces of information $S_1, S_2, ..., S_n$ so that:

- Any t of the S_i can be used to access/construct S.
- No $t-1$ of them is enough to access/construct S.

Such a scheme is called a **secret sharing scheme** of **size** n and **threshold** t. We might also like it to be the case that knowing only $t-1$ of the S_i tells you **nothing** about S. Such schemes are called **perfect**.

Example 7.1. There's an obvious way of creating perfect secret sharing schemes of size n and threshold $t = n$. Let $M \geq 1$ and suppose we wish to share secret value $S \in \mathbb{Z}/M\mathbb{Z}$. We choose $n-1$ random values $S_1, S_2, ..., S_{n-1}$ and let $S_n \equiv S - (S_1 + ... + S_{n-1}) \bmod m$.

The secret S can be reconstructed using all n of the S_i since their sum is $S \bmod M$. Knowing $n-1$ of the values doesn't tell us anything about S, since the remaining value can take multiple values, giving multiple values for S.

This scheme is perfect since the the remaining value is equally likely to be any element of $\mathbb{Z}/M\mathbb{Z}$, and running through all such choices gives all possible values of S (so we haven't learned anything about S).

The main issue with having threshold $t = n$ is that everyone has to be present in order to reconstruct the secret. This seems like overkill and so we might seek schemes with threshold $t < n$. We'll see an example of this soon.

Lemma 7.2. *Let K be a field and $x_1, x_2, ..., x_t \in K$. Then:*

$$det \begin{pmatrix} 1 & x_1 & x_1^2 & \dots & x_1^{t-1} \\ 1 & x_2 & x_2^2 & \dots & x_2^{t-1} \\ 1 & x_3 & x_3^2 & \dots & x_3^{t-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_t, & x_t^2 & \dots & x_t^{t-1} \end{pmatrix} = \prod_{1 \leq i < j \leq t} (x_j - x_i).$$

Proof. Recall Leibniz's formula for determinants:

$$\det(a_{i,j}) = \sum_{\sigma \in S_t} \operatorname{sgn}(\sigma) \prod_{i=1}^{t} a_{i,\sigma(i)}.$$

Treating the x_i as variables it is now clear that the determinant of our matrix is a polynomial $d \in K[x_1, x_2, ..., x_t]$ of degree $\frac{t(t-1)}{2}$.

Note that setting $x_i = x_j$ for any $i \neq j$ gives determinant 0, since two rows of the matrix would be equal. Thus $(x_j - x_i)$ divides d for each $i \neq j$ and so:

$$d = d' \prod_{1 \leq i < j \leq t} (x_j - x_i),$$

for some polynomial $d' \in K[x_1, x_2, ..., x_t]$. However, the product also has degree $\frac{t(t-1)}{2}$ and so d' must be constant. Further, $d' = 1$ since the diagonal term of the determinant is $x_2 x_3^2 ... x_t^{t-1}$ and this term appears in the product with coefficient 1 (take the product of the left hand entries of the brackets). □

A matrix of the above form is called a **Vandermonde matrix**. The fact that the determinant of such a matrix takes a nice form allows us to prove the following.

Proposition 7.3. *Let K be a field and $P_1, ..., P_t \in K^2$ be points with distinct x-coordinates. Then there is a unique polynomial $f \in K[x]$ of degree $t-1$ such that $f(x_i) = y_i$ (with $P_i = (x_i, y_i)$).*

Proof. Let $f = a_0 + a_1 x + a_2 x^2 + ... + a_{t-1} x^{t-1}$. In order to satisfy $f(x_i) = y_i$ we must solve the equations:

$$\begin{aligned} a_0 + a_1 x_1 + a_2 x_1^2 + ... + a_{t-1} x_1^{t-1} &= y_1 \\ a_0 + a_1 x_2 + a_2 x_2^2 + ... + a_{t-1} x_2^{t-1} &= y_2 \\ \vdots \qquad\qquad \vdots \qquad\qquad & \vdots \\ a_0 + a_1 x_t + a_2 x_t^2 + ... + a_{t-1} x_t^{t-1} &= y_t \end{aligned}$$

This linear system can be written as $A\mathbf{z} = \mathbf{y}$ with A a Vandermonde matrix, $\mathbf{z} = (a_0, a_1, ..., a_{t-1})^T$ and $\mathbf{y} = (y_1, y_2, ..., y_t)^T$. This system has a unique solution, since $\det(A) = \prod_{1 \leq i < j \leq t} (x_j - x_i) \neq 0$ (since the x_i are distinct). □

The above proposition is a generalisation of the fact that **two** points in the plane describe a unique **line**, **three** points determine a unique **parabola**, etc. Note that the above proof actually gives a method for finding the polynomial f, we solve a linear system of equations.

Example 7.4. If $t = 2$ this system of equations becomes:

$$\begin{pmatrix} 1 & x_1 \\ 1 & x_2 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}.$$

Solving gives:

$$\begin{pmatrix} a_0 \\ a_1 \end{pmatrix} = \frac{1}{x_2 - x_1} \begin{pmatrix} x_2 & -x_1 \\ -1 & 1 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \frac{1}{x_2 - x_1} \begin{pmatrix} x_2 y_1 - x_1 y_2 \\ y_2 - y_1 \end{pmatrix},$$

so that the line through points $(x_1, y_1), (x_2, y_2)$ (with $x_1 \neq x_2$) has equation:

$$y = \frac{x_2 y_1 - x_1 y_2}{x_2 - x_1} + \frac{y_2 - y_1}{x_2 - x_1} x.$$

This agrees with what you already know (e.g. the x coefficient is the gradient).

Lagrange came up with a much **quicker** way of constructing the polynomial passing through t points.

Theorem 7.5. *(Lagrange Interpolation) The Let K be a field and $P_1, ..., P_t \in K^2$ be points with distinct x-coordinates. The unique polynomial $f \in K[x]$ of degree $t - 1$ in Proposition 7.3 is given by:*

$$f(x) = \sum_{i=1}^{t} y_i \prod_{j \neq i} \frac{x - x_j}{x_i - x_j}.$$

Proof. Proposition 7.3 proved uniqueness and so we only need to show that $f(x_i) = y_i$. This is clear since:

$$f(x_i) = y_i \prod_{j \neq i} \frac{x_i - x_j}{x_i - x_j} + \sum_{a \neq i} y_a \prod_{j \neq a} \frac{x_i - x_j}{x_a - x_j} = y_i.$$

(The first product term is 1 by cancellation, and each of the second product terms is 0, since if $a \neq i$ then the numerator contains a term with $j = i$, i.e. $(x_i - x_j) = 0$). □

Example 7.6. The line going through $(x_1, y_1), (x_2, y_2) \in K^2$ (with $x_1 \neq x_2$) is:

$$y = y_1 \frac{x - x_2}{x_1 - x_2} + y_2 \frac{x - x_1}{x_2 - x_1} = \frac{x_2 y_1 - x_1 y_2}{x_2 - x_1} + \frac{y_2 - y_1}{x_2 - x_1} x.$$

The parabola going through $(1, 4), (2, 6)$ and $(-1, 10)$ in $\mathbb{R}^2$ is:

$$\begin{aligned} y &= 4\frac{(x-2)(x+1)}{(1-2)(1+1)} + 6\frac{(x-1)(x+1)}{(2-1)(2+1)} + 10\frac{(x-1)(x-2)}{(-1-1)(-1-2)} \\ &= \frac{5}{3}x^2 - 3x + \frac{16}{3}. \end{aligned}$$

We can now see how to create a perfect secret sharing scheme of size n and threshold $t \leq n$.

Shamir's secret sharing scheme

1. Alice views her secret as $S \in \mathbb{F}_p$ for some prime $p > n$ and chooses a random polynomial of the form $f(x) = S + a_1x + a_2x^2 + ... + a_{t-1}x^{t-1} \in \mathbb{F}_p$. She then chooses n random distinct values $x_1, x_2, ..., x_n \in \mathbb{F}_p$ and distributes the points $P_i = (x_i, f(x_i))$ to the n people.
2. Any t people can then reconstruct S by using Lagrange Interpolation to find f and read off the constant term.
3. No $t-1$ people can reconstruct S uniquely since there is more than one degree $t-1$ polynomial passing through $t-1$ of the P_i (the corresponding system of equations doesn't have a unique solution).

Since we are working over $\mathbb{F}_p$ the above scheme is actually perfect. Knowledge of only $t-1$ of the P_i restricts the space of potential polynomials to an **affine space** of dimension at least 1 with **all** constant terms appearing among these polynomials equally (so that you learn nothing about S).

Example 7.7. Alice wants to share $S = 5 \in \mathbb{F}_{11}$ between $n = 4$ people with threshold $t = 3$. She chooses the polynomial $f(x) = 5 + 7x + 4x^2$ and creates the points $P_1 = (1, 5), P_2 = (2, 2), P_3 = (5, 8), P_4 = (7, 8)$.

Applying Lagrange Interpolation to any triple, say P_1, P_2, P_3 gives secret $S = 5$:

$$\begin{aligned} f(x) &= 5\frac{(x-2)(x-5)}{(1-2)(1-5)} + 2\frac{(x-1)(x-5)}{(2-1)(2-5)} + 8\frac{(x-1)(x-2)}{(5-1)(5-2)} \\ &= \mathbf{5} + 7x + 4x^2. \end{aligned}$$

The general quadratic passing through P_1 and P_2 say has the form:

$$\alpha + (4\alpha + 9)x + (6\alpha + 7)x^2,$$

for $\alpha \in \mathbb{F}_{11}$. The constant term is arbitrary and so nothing is learned about S.

These lecture notes survey a lot of the major schemes in modern cryptography, but we have only just scratched the surface. However, hopefully in reading these notes you have become interested in the emerging world of mathematical cryptography.

In other words, I hope you **enjoyed** the course...