CCS Discrete Math I | Professor: Padraic Bartlett

Lecture 5: Cryptography and Groups

Week 5 | *UCSB 2014*

1 Key Exchanges: A Puzzle

Before we start, consider the following puzzle:

Puzzle. The Untrustworthy Ferryman.

- Good news! You've found yourself in the possession of a shoebox full of rubies, diamonds, gold, etc. You also have a padlock and key, which you can use to lock the outside of the box.
- Bad news! You're trapped on a desert island in the middle of nowhere.
- Good news! Every day at noon, a ferryman approaches your island. His boat is just big enough to contain him and a small box. If you could get the box delivered to your friend on the mainland, in a way where he could open the box, they could use a handful of the jewels to rescue you and set you free!
- Bad news! The ferryman will steal and not deliver anything that is not a locked box. Locked boxes he will take and deliver to your friend on the mainland for the price of one jewel. But anything else he will steal!
- Possibly useful news? Your friend also has a box and padlock and key, and the ferryman is willing to take packages back to you from your friendas part of the deal. But again, he will steal anything that is not a locked box.

How can you get him the treasure?

The answer is on the next page; try to solve it before continuing!

Answer. Here's the answer!

- You lock up your box and send it to your friend via the ferryman. Because the box is locked, the ferryman is honest and delivers the box.
- Your friend receives the box, and **locks the box with his padlock as well**! I.e. now the box has two locks on it.
- The doubly-locked box is sent back to you. You remove your lock, and send it back to your friend.
- Your friend receives the box with just his lock on it! He removes his lock, takes the jewels, rescues you, and you live your life in comfort and/or splendor.

This idea, roughly speaking, is how modern cryptographic systems work!

2 Public Key Exchange Systems

In general, a public key exchange system, roughly speaking, is any cryptographic system that works like our puzzle above — i.e. where there are two parties that are communicating, who do so in the following manner:

- Party A takes a message and encrypts it with a secret key, known only to them. Then they send this message to party B.
- Party B takes that encrypted message and encrypts it a second time with their secret key, and sends it back to A
- Party A then takes this doubly-encrypted message and decrypts it with their key. If our encryption and decryption functions **commute** — in other words, if encrypting and then decrypting is the same thing as decrypting and then encrypting — then this "undoes" the first layer of encryption, leaving a message encrypted with only B's key. This is sent back to B.
- Party B then takes that message and decrypts it; it is now readable by party B!

This is the idea for how most modern cryptographic systems work. With the rest of this class, we're going to explore one specific method of this encryption scheme: the **Diffie-Hellman** key exchange system!

To do this, we need a few techniques involving groups:

3 Group Theorems

Here's a fun property:

Theorem 1. ***Fermat's Little Theorem.*** *Let p be a prime number. Take any $a \neq 0$ in $\mathbb{Z}/p\mathbb{Z}$. Then*

$$a^{p-1} \equiv 1 \mod p.$$

Proof. There are many different proofs of this remarkable theorem! The one we present here is perhaps one of the shortest, and relies on the **binomial theorem**:

Theorem. For any positive integer n, and any x, y, we have

$$(x+y)^n = \sum_{i=0}^{n} \binom{n}{i} x^{n-i} y^i.$$

We use this theorem to prove Fermat's little theorem by induction on a.
Our base case is easy: when $a = 1$, we have

$$1^{p-1} = 1 \equiv 1 \mod p.$$

Induction is not too much harder. We start by assuming that

$$k^{p-1} \equiv 1 \mod p,$$

and now want to prove that

$$(k+1)^{p-1} \equiv 1 \mod p.$$

To do this, simply look at the quantity

$$(k+1)^p.$$

By the binomial theorem, we have

$$\begin{aligned}(k+1)^p = \sum_{i=0}^{p} \binom{p}{i} k^{p-i} 1^i &= k^p + \binom{p}{1} k^{p-1} 1^1 + \binom{p}{2} k^{p-2} 1^2 + \ldots + \binom{p}{p-1} k^1 1^{p-1} + \binom{p}{p} 1^p \\ &= k^n + \binom{p}{1} k^{p-1} + \binom{p}{2} k^{p-2} + \ldots + \binom{p}{p-1} k + 1.\end{aligned}$$

Now, notice the following property:

Observation. If p is prime, then for any $1 \leq i \leq p-1$, the quantity

$$\binom{p}{i} = \frac{p!}{i! \cdot (p-i)!} = \frac{(p-i+1)(p-i+2)\ldots(p)}{i!}$$

is an integer multiple of p. This is because

- p is a factor of the numerator, and
- no factors of p are in the denominator, because p is prime and greater than i.

For example,

$$\binom{5}{3} = \frac{5!}{3! \cdot (2)!} = \frac{(3)(4)(5)}{3!} = 10,$$

which is a multiple of 5.

Given this observation, we can now see that

$$\begin{aligned}(k+1)^p &= k^p + \binom{p}{1}k^{p-1} + \binom{p}{2}k^{p-2} + \ldots + \binom{p}{p-1}k + 1 \\ &\equiv k^p + 0 \cdot k^{p-1} + \ldots + 0 \cdot k + 1 \mod p \\ &\equiv k^p + 1 \mod p.\end{aligned}$$

This is because all of the $\binom{p}{i}$ terms are multiples of p, and therefore congruent to 0 mod p.

Now, we use our inductive hypothesis, which tells us that

$$\begin{aligned}k^{p-1} &\equiv 1 \mod p \\ \Rightarrow k^p &\equiv k \mod p;\end{aligned}$$

plugging this in gives us that

$$(k+1)^p \equiv k+1 \mod p.$$

Dividing through by $(k+1)$ yields

$$(k+1)^{p-1} \equiv 1 \mod p,$$

as claimed.

□

It's worth noting that this theorem generalizes in a powerful fashion to something that is true for **all** groups:

Theorem. Let $\langle G, \cdot \rangle$ be a finite group, and $g \in G$ be any element of G. Define the **order** of g to be the smallest value of n such that $g^n = id$. Then the order of g always divides the total number of elements in G, $|G|$.

Before we prove this claim, we introduce a few group-theory concepts! First, consider the notion of a **subgroup**:

Definition. Take any group $\langle G, \cdot \rangle$, and a subset $H \subseteq G$. We say that H is a **subgroup** of G if the following three properties hold:

1. **Closure**: For any two elements $h_1, h_2 \in H$, $h_1 \cdot h_2$ is also in H.
2. **Identity**: The identity id of G is in H.
3. **Inverses**: If $h_1 \in H$, then h_1^{-1} is also in H.

Notice that any subgroup of a group is a group in its own right! This is because the associativity property is "inherited" from the larger group, and we've already checked all of the other properties for being a group.

Examples. The set of even numbers is a subgroup of the integers under addition! This is because 0 is an even number (identity), the sum of any two even numbers is even (closure), and the additive inverse of any even number is also even (inverses.)

Conversely, the set of all odd numbers is not a subgroup of the integers under addition: because 0 is not odd, this subset does not satisfy the identity property!

The set of all powers of 2, $\{2^n \mid n \in \mathbb{Z}\}$ is a subgroup of the nonzero real numbers under multiplication! This is because $1 = 2^0$ (identity,) $2^n \times 2^m = 2^{n+m}$ (closure), and $2^n \cdot 2^{-n} = 2^0 = 1$ (inverses) all hold for this subgroup.

Conversely, the set of all even integers is not a subgroup of the nonzero reals under multiplication, as there is no multiplicative inverse for 2 in this set!

The set of all rotations, along with the identity is a subgroup of the group D_{2n} of symmetries of a n-gon: to see this, notice that we have the identity by definition, the inverse of rotating by θ is another rotation (by $2\pi - \theta$), and the composition of two rotations (by θ, φ) is just the rotation given by summing their angles ($\theta + \varphi$.)

Example. There is one particularly useful type of subgroup that comes up often. Take any group $\langle G, \cdot \rangle$, and any element $a \in G$. Look at the subset

$$\{a^k \mid k \in \mathbb{Z}\},$$

where we let $a^0 = id$, $a^k = \overbrace{a \cdot a \cdot \ldots \cdot a}^{k \text{ times}}$ for positive k, and $a^{-k} = \overbrace{a^{-1} \cdot a^{-1} \cdot \ldots \cdot a^{-1}}^{k \text{ times}}$ for negative k.

I claim this is a subgroup of G. To see why, simply note that we have identity by definition, and that for any a^k, $a^k \cdot a^l = a^{k+l}$ is still in our group, and $a^k \cdot a^{-k} = id$. So we have inverses and closure!

This group is particularly useful; we call it the **subgroup generated by** a, and denote it as $\langle a \rangle$ Notice the following property:

Proposition. The number of elements in $\langle a \rangle$ is the **order** of a.

Proof. This is because if $a^n = id$, then we have the following:

$$a^k = a^l, \forall k \equiv l \mod n.$$

This is because we can simply multiply in copies of a^n, a^{-n} to change the exponent by multiples of n, without changing anything (because $a^n = id = a^{-n}$!) Therefore, the only possibly distinct elements of $\langle a \rangle$ are powers of a with distinct exponents mod n: this tells us that $\langle a \rangle$ has at most as many elements as the order of a!

To see equality, suppose that the order of a was n, but $\langle a \rangle$ was something less than n. This means that among the n powers of a with distinct exponents mod n — explicitly, $a^0, a^1, \ldots a^{n-1}$ — there are two that are equal! Consequently, we have $a^k = a^l$ for some $k \neq l$ in $\{0, \ldots n-1\}$; if we assume that k is the smaller one of the two powers and multiply by a^{-k} on both sides, we get $a^{l-k} = id$, for some positive integer $l - k$ strictly less than n. This contradicts our assumption that the order of a was n! Therefore we have equality, as claimed. □

We need one more concept to prove our claim:

Definition. Suppose that $\langle G, \cdot \rangle$ is a group, $s \in G$ is some element of G, and H is a subgroup of G. We define the **right coset** of H corresponding to s as the set

$$Hs = \{hs \mid h \in H\}.$$

We will often omit the "right" part of this definition and simply call these objects cosets.

Example. Consider the group $G = \langle \mathbb{Z}, + \rangle$. One subgroup of this group is the collection of all multiples of 5: i.e.

$$H = \{\ldots -15, -10, -5, 0, 5, 10, 15 \ldots\}$$

This subgroup has several cosets:

- $s = 0$: this forms the coset

  $$H + 0 = \{\ldots -15, -10, -5, 0, 5, 10, 15 \ldots\},$$

  which is just H itself.
- $s = 1$: this forms the coset

  $$H + 1 = \{\ldots -14, -9, -4, 1, 6, 11, 16 \ldots\}.$$

- $s = 2$: this forms the coset

  $$H + 2 = \{\ldots -13, -8, -3, 2, 7, 12, 17 \ldots\}.$$

- $s = 3$: this forms the coset

  $$H + 3 = \{\ldots -12, -7, -2, 3, 8, 13, 18 \ldots\}.$$

- $s = 4$: this forms the coset

  $$H + 4 = \{\ldots -11, -6, -1, 4, 9, 14, 19 \ldots\}.$$

Notice that this collection of cosets above is indeed the collection of **all** of the possible cosets of H within G: if we take any other element in $\mathbb{Z}$, like say 13, we'll get one of the five cosets above: i.e.

$$H + 13 = \{\ldots -2, 3, 8, 13, 18 \ldots\} = H + 3.$$

In general, $H + x = H + y$ for any $x \equiv y \mod 5$.

Example. Consider the group $G = \langle (\mathbb{Z}/7\mathbb{Z})^\times, \cdot \rangle$, i.e. the nonzero integers mod 7 with respect to the multiplication operation. This has the set

$$H = \{1, 6\}$$

as a subgroup (check this if you don't see why!)

This group has the following cosets:

- $s = 1$, which creates the cosets $H \cdot 1 = H$,
- $s = 2$, which creates the coset

$$H \cdot 2 = \{2, 5\}.$$

- $s = 3$, which creates the coset

$$H \cdot 3 = \{3, 4\}.$$

- $s = 4$, which creates the coset

$$H \cdot 4 = \{4, 3\}.$$

  Notice that this coset is the same as $H \cdot 3$.

- $s = 5$, which creates the coset

$$H \cdot 5 = \{5, 2\}.$$

  Notice that this coset is the same as $H \cdot 2$.

- $s = 6$, which creates the coset

$$H \cdot 6 = \{6, 1\}.$$

  Notice that this coset is the same as H.

Example. Consider the group S_3. This group has the subgroup

$$H = \{id, (123), (132)\}$$

as a subgroup. This subgroup has two possible distinct cosets:

- $H \cdot id = H \cdot (123) = H \cdot (132)$ are all the same coset, which is just H.
- $H \cdot (12) = H \cdot (13) = H \cdot (23) = \{(12), (13), (23)\}$.

Cosets have a number of remarkably useful properties:

Theorem. Take any finite group $\langle G, \cdot \rangle$ and any subgroup H.

1. For any $s \in G$, the right coset Hs is equal to H if and only if $s \in H$.
2. Two cosets Hs, Ht are either completely identical or completely disjoint.
3. The various possible cosets of H **partition** G into a collection of disjoint subsets. (In particular, this proves that the number of elements in H must divide the number of elements in G.)
4. All cosets of H are the same size: i.e. $|Hs| = |H|$ for any $s \in G$.

Proof. Take any group $\langle G, \cdot \rangle$ and any subgroup H. We prove these properties in order.

For (1): take any $s \in G$. If $s \in H$, then because H is a subgroup we have that s^{-1} is in H, and therefore that for any $h \in H$ we have hs^{-1} in H. Consequently, every member of H is in Hs. As well, any member of Hs is a product of two elements of H; therefore by closure, Hs is a subset of H. Therefore we have proven that $Hs = H$, as claimed.

Conversely, when $s \notin H$, we want to show that $Hs \neq H$. But this is trivial: because H contains the identity, we know that $s = s \cdot id \in Hs$, but $s \notin H$; so these are different sets.

For (2): Take any two cosets Hs, Ht. If they are not completely disjoint, then there is some element they share in common: in other words, there is some $h_1, h_2 \in H$ such that $h_1 s = h_2 t$. So: take any $h_3 s \in Hs$. Because $h_3 s = h_3 h_1^{-1} h_1 s = h_3 h_1^{-1} h_2 t \in Ht$, we have just shown that any element of Hs is in Ht! Similarly, we can take any $h_3 t \in Ht$, and see that $h_3 t = h_3 h_2^{-1} h_2 t = h_3 h_2^{-1} h_1 s \in Hs$, and therefore see that every element of Ht is in Hs. Therefore, if two cosets share any elements in common they are identical, as claimed.

For (3): Simply notice that for any element $g \in G$, Hg contains g; this is because the identity is in H, and therefore that $id \cdot g = g \in Hg$. But this means that every element is in some coset: if we combine this with (2), we have that every element is in exactly one coset. But this is what it means to be a partition!

For (4): on one hand, we know that $Hs = \{hs \mid h \in H\}$ means that we cannot have more elements in Hs than we have in H, because we only get one element in Hs for each element of H. But if we ever had $h_1 s = h_2 s$, we could multiply by s^{-1} on the right to get $h_1 = h_2$: consequently, we can conclude that we have exactly as many elements in Hs as we do in H! □

With all of this machinery in place, our proof of the generalized Fermat's Little Theorem is remarkably easy:

Theorem. Let $\langle G, \cdot \rangle$ be a finite group, and $g \in G$ be any element of G. Define the **order** of g to be the smallest value of n such that $g^n = id$. Then the order of g always divides the total number of elements in G, $|G|$.

Proof. Look at $\langle g \rangle$. This is a subgroup of size n, where n is the order of g, as discussed earlier.

Look at all of the cosets of $\langle g \rangle$. These cosets partition G into disjoint sets, all of size n, as proven above. Therefore we have that $|G| = n \cdot$ (number of cosets); consequently n divides $|G|$, as claimed! □

This actually has a trivial strengthening: instead of looking at $\langle a \rangle$, we could simply use the proof above on **any** subgroup! This gives us Lagrange's theorem:

Theorem. Let $\langle G, \cdot \rangle$ be a finite group, and H be any subgroup of G. Then the order of H divides the order of G.

Why do we care about this? Because it lets us create one of the first (and one of the most fundamental) modern cryptographical systems!

Before we go into modern systems, however, let's wrap our heads around cryptography with some classical examples:

4 Cryptography: Classical Approaches

Definition. An **encryption algorithm**, or **cipher**, is a method that allows us to turn normal, plainly-readable text into difficult-to-read **ciphertext**, via the use of a secret key. Formally, we write

$$enc(k, \text{plaintext}) = \text{ciphertext},$$
$$dec(k, \text{ciphertext}) = \text{plaintext},$$

where

- plaintext is a message we want to encrypt,
- k is a key,
- enc is a function that takes in a unencrypted message and a key, and outputs an encrypted version of that message, and
- dec is a function that takes in an encrypted message and a key, and outputs the unencrypted version of that message.

In this system, the functions enc, dec are assumed to be widely known. There are cryptographic systems that do not assume this, but they are widely considered to be "bad." There are a number of reasons for why this is believed to be true (see security through obscurity and Kerckhoffs's principle for a wider discussion of this philosophy;) one of the simpler arguments is that there are many more ways to determine what an algorithm does (intimidate/bribe any of a number of engineers, steal a copy of the code, pose as a legitimate user and buy the algorithm, etc.) than to steal a specific key (which can only be done by attacking the specific user of the key you want.)

Typically, in cryptography, we refer to the two communicating parties as Alice and Bob (A and B for short, but seriously everyone uses Alice and Bob,) and the hypothetical third-party eavesdropper as Eve (E.) Given any encryption system, we typically evaluate its strength by looking at it in the following three situations:

1. The attacker, Eve, has access to a number of cipher texts.
2. The attacker, Eve, has access to a number of cipher texts and their original plaintexts.
3. The attacker, Eve, has the ability to generate cipher texts for whatever plaintext inputs they choose.

The first situation is the most common one: it is typically assumed that the attacker Eve has access to most, if not all, of the encrypted traffic that Alice and Bob send back and forth to each other. The second is stronger, but not unreasonable to expect; in many situations (see: WWII and the Enigma machine for some great stories) the attacker will be able to "steal" some unencrypted messages via subterfuge, and it would be great if an encryption method could still work even with this occasionally happening. The third is stronger yet: it basically is a kind of system that can withstand most anything, as long as the keys remain hidden. Strong cryptographical systems work in all of these systems, and are what we want to find.

People have been creating encryption schemes for thousands of years; essentially, as long as there have been people with secrets to keep, there have been ways to keep things secret. We study several of these here:

Algorithm. The Caesar-shift cipher. The Caesar-shift cipher, whose first recorded use was by Julius Caesar to protect various military secrets, is the following encryption scheme. Given a plaintext message m and a key k, "encrypt" m by doing the following: one-by-one, take each character of m and circularly shift it over k places to the right in the alphabet. Caesar historically used this cipher with a shift of three: i.e. $A \mapsto D$, $B \mapsto E$, $\ldots W \mapsto Z$, $X \mapsto A$, $Y \mapsto B$, $Z \mapsto C$. The decryption scheme is similar: take your encrypted message and character-by-character, circularly shift each letter over k places to the left in the alphabet.

This cipher, with a key of 13, is known as "ROT13" and is frequently used on the Internet to hide spoilers; this cipher-key combo is particularly convenient, because its encryption and decryption functions are identical (shifting right by 13 and left by 13 are the same in a 26-character alphabet.)

Example. Take the message[1]

```
"Just the place for a Snark!" the Bellman cried,
As he landed his crew with care;
Supporting each man on the top of the tide
By a finger entwined in his hair.
```

If we applied a Caesar shift with key 4, we would get the message

```
"Nywx xli tpegi jsv e Wrevo!" xli Fippqer gvmih,
Ew li perhih lmw gvia amxl gevi;
Wyttsvxmrk iegl qer sr xli xst sj xli xmhi
Fc e jmrkiv irxamrih mr lmw lemv.
```

Weaknesses. This is a very weak cipher. In particular, it is easy to beat with brute-force approaches: for example, suppose we saw the text

```
Ns ymj gjlnssnsl ymjwj bfx stymnsl, bmnhm jcuqtiji.
```

we could simply just go through values of k until we got something that looked like a promising translation:

```
Mr xli fikmrrmrk xlivi aew rsxlmrk, almgl ibtpshih.
Lq wkh ehjlqqlqj wkhuh zdv qrwklqj, zklfk hasorghg.
Kp vjg dgikppkpi vjgtg ycu pqvjkpi, yjkej gzrnqfgf.
Jo uif cfhjoojoh uifsf xbt opuijoh, xijdi fyqmpefe.
In the beginning there was nothing, which exploded.
```

Given that there are only 26 values of k to pick, this should be something we can do relatively quickly. Moreover, we could just do this to a small sample of text if it took us too long to translate everything, as there's usually only one shift that's going to make our text look readable.

[1] From **The Hunting of the Snark**, by Lewis Carroll. It's pretty great.

Algorithm. Simple substition ciphers. One key issue with the algorithm above was that the range of possible key choices was far too small: we could simply adopt a brute-force approach and look at all possible outputs of our decryption function under different keys, and discover the original plaintext in this way.

A solution to this was the idea of a simple substition cipher, which is defined as follows: first, write down the alphabet. Then, write down some permutation ρ of that alphabet: i.e.

ABCDEFGHIJKLMNOPQRSTUVWXYZ

This permutation ρ is the key for an encryption scheme defined as follows: take a plaintext message, and character-by-character replace each letter in the plaintext message with a character from the permutation. For example, if we used the permutation described above, we would have $A \mapsto E, B \mapsto J, C \mapsto W, D \mapsto D, \ldots$

This algorithm avoids the weakness we've noticed that Caesar-shift is weak to: where the Caesar shift only had 26 keys, this algorithm has as many keys as there are ways to permute the characters of the alphabet. If we count, we can see that there are 26! ways in which to do this: this is because in creating a permutation we are choosing one of our 26 characters for A to map to, any of the remaining 25 characters for B to map to, and so on/so forth until we have have one last character for Z to map to.

26! is a much larger search space than 26: it's roughly $4 \cdot 10^{26}$. By comparison, the fastest supercomputer on record (as of November, 2013, and as far as I know) can perform roughly $34 * 10^{15}$ calculations per second; if it could check whether one given permutation was a viable interpretation of our encrypted text per calculation, it would require about 373 years to test all possible calculations. So, at the least, brute-force is not always the *best* strategy to use...

Example. Under the permutation

ABCDEFGHIJKLMNOPQRSTUVWXYZ

the phrase

is transformed into the phrase

Weaknesses. While this algorithm is pretty much immune to brute-force attacks, it is still remarkably easy to crack, even by hand. We can do this by studying the underlying structure of the plaintext message sent — i.e. the structure of the English language itself! — and use this information to crack the algorithm.

To get an idea of how this would work, consider the following longer sample of ciphertext:

```
Ony rgon af ony tdznoyapk hgb dk qykyo ab gii kdjyk qu ony
dbycpdodyk af ony kyifdkn gbj ony outgbbu af yedi hyb.
Qiykkyj dk ny wna, db ony bghy af vngtdou gbj zaaj wdii,
knyrnytjk ony wygl ontapzn ony egiiyu af ony jgtlbykk, fat ny
dk otpiu ndk qtaonyt'k lyyryt gbj ony fdbjyt af iako vndijtyb.
Gbj D wdii kotdly jawb prab onyy wdon ztygo eybzygbvy gbj
fptdapk gbzyt onaky wna gooyhro oa radkab gbj jykotau Hu
qtaonytk.  Gbj uap wdii lbaw D gh ony Iatj wnyb D igu Hu
eybzygbvy prab uap.
```

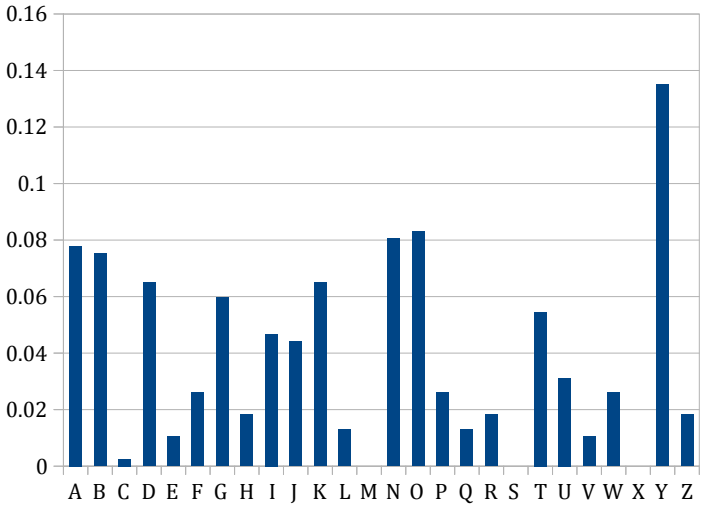
On its surface, this doesn't look much like English anymore. However, there are still bits of underlying structure that we can use to figure out what the cipher might be! For example, you could make the following observations:

- There is only one single-letter word that occurs in our passage above, the word "d." In English, there are only two one-letter words: "a" and "I." Consequently, it would seem rather likely that one of the letters A or I were mapped to D.
- Similarly, there is a character "k" that occurs as a single character after an apostrophe in our text sample; this likely eliminates every character except for "s", "t" or rarely "d." So we likely have that one of S or T maps to D.
- The three-letter word "ony" repeatedly occurs. If you do a word frequency analysis of the English language, you can see that the two most common three-letter words by a decent margin are "the" and "and"; so we could try seeing what happens if we assume one of these two words map to ONY.
- ...

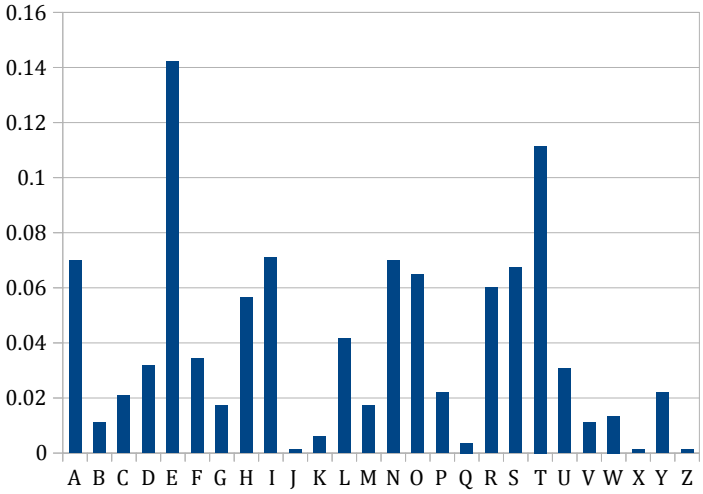
An objection you could make here is that we're deriving all of this structure from the "unencrypted" parts of our communication – i.e. we're figuring things out about our message by looking at the spaces and punctuation, which weren't encrypted!

In practice, people sometimes deal with this specific attack by simply omitting punctuation and spaces (or replacing all instances of a space with an infrequently-used letter like "q"). However, this doesn't stop us from a more fundamental method of attack — character frequency!

Specifically: notice that the characters in the above text occur with the following frequencies:



Now, notice that if you were to take a sufficiently large sample of works in English — say, a handful of Iain M. Banks novels, or the works of Raymond Chandler — you also have certain character frequencies! Intuitively, this makes sense: we're much more likely to see the letter "a" than the letter "q," for example. In general, English text tends to have the following character distributions:



This gives us the following observations:

- Given the above two graphs, we might assume that the most frequently-occurring character in our sample set, "y", corresponds to the most frequently-occurring character in English, "e."

- Consequently, if we looked through our text and picked out the most frequently-occurring three-letter cipherphrase, "ony", we could assume that this matches up with frequently-occurring three-letter objects in English! Again, studying a large body of work reveals that the most frequently occurring trigrams in English, in order, are

1. the	5. ing	9. nde	13. tis
2. and	6. ion	10. has	14. oft
3. tha	7. tio	11. nce	15. sth
4. ent	8. for	12. edt	16. men

  Of these, "the" is the only one in the top 8 that ends in "e": so it may be reasonable to assume that "the" and "ony" correspond to each other!

- If we apply this observation — that it is likely that $T \mapsto O, H \mapsto N, E \mapsto Y$ — we get

```
The rgth af the tdzhteapk hgb dk qeket ab gii kdjek qu the
dbecpdtdek af the keifdkh gbj the tutgbbu af eedi heb.
Qiekkej dk he wha, db the bghe af vhgtdtu gbj zaaj wdii,
kherhetjk the wegl thtapzh the egiieu af the jgtlbekk, fat he
dk ttpiu hdk qtathet'k leeret gbj the fdbjet af iakt vhdijteb.
Gbj D wdii kttdle jawb prab thee wdth ztegt eebzegbve gbj
fptdapk gbzet thake wha gttehrt ta radkab gbj jekttau Hu
qtathetk.  Gbj uap wdii lbaw D gh the Iatj wheb D igu Hu
eebzygbvy prab uap.
```

  The symbols we believe to be "correct" are in red.

- We now return to our earlier observations about the text that used its structure: (1) that the apostrophe character "k" likely corresponds to one of "s" or "t", and (2) that the only occurring single character "d" should correspond to one of the two English single-letter words, "I" or "a". In particular, the letter "d" is always capitalized in our text, so "I" likely maps to "D"! Furthermore, if we're assuming that "t" maps to "o", then we cannot have it mapping to "k" as well: therefore, we likely have "s" mapping to "k". We can further update our text here with these observations:

```
The rgth af the tizhteaps hgb is qeset ab gii sijes qu the
ibecpities af the seifish gbj the tutgbbu af eeii heb.
Qiessej is he wha, ib the bghe af vhgtitu gbj zaaj wiii,
sherhetjs the wegl thtapzh the egiieu af the jgtlbess, fat he
is ttpiu his qtathet's leeret gbj the fibjet af iast vhiijteb.
Gbj I wiii sttile jawb prab thee with ztegt eebzegbve gbj
fptiaps gbzet thase wha gttehrt ta raisab gbj jesttau Hu
qtathets.  Gbj uap wiii lbaw I gh the Iatj wheb I igu Hu
eebzygbvy prab uap.
```

From here, we can simply look at the phrases with all but one character determined — "qeset," "sijes," "with" — and make more guesses at characters. I.e. the only common word word ending in "eset" with five characters is "beset," so we have b mapping to q; similarly, "si?es" is either sides, sires, sites, sixes or sizes. We can eliminate sites on the grounds that we are already using the character t, and sixes and sizes on the fact that the character j occurs often in our sample text, and it's unlikely that x or z would do so. So we're down to j correspond to either d or r: the high frequency of the three-letter phrase "gbj" would lead us to believe that this is not "r," as there are not many three-letter words ending in "r" that we'd expect to see this many times, while there definitely are such words (specifically, "and") that end in d and occur this frequently!

In turn this motivates us to guess that the rest of gbj actually corresponds to "and:" if we do this, we get

```
The rath af the tizhteaps han is beset an aii sides bu the
inecpities af the seifish and the tutannu af eeii hen.
Biessed is he wha, in the nahe af vhatitu and zaad wiii,
sherhetds the weal thtapzh the eaiieu af the datlness, fat he
is ttpiu his btathet's leeret and the findet af iast vhiidten.
And I wiii sttile dawn pran thee with zteat eenzeanve and
fptiaps anzet thase wha attehrt ta raisan and desttau Hu
btathets.  And uap wiii lnaw I ah the Iatd when I iau Hu
eenzyanvy pran uap.
```

Repeating this process one more time — look for words like "B?essed," "da??ness," "ine??ities," etc. that are mostly completed, compare to the English language, make a guess — eventually gives us that our permutation is

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

and that our corresponding text is the following passage:

```
The path of the righteous man is beset on all sides by the
inequities of the selfish and the tyranny of evil men.
Blessed is he who, in the name of charity and good will,
shepherds the weak through the valley of the darkness, for he
is truly his brother's keeper and the finder of lost children.
And I will strike down upon thee with great vengeance and
furious anger those who attempt to poison and destroy My
brothers.  And you will know I am the Lord when I lay My
vengeance upon you.
```

Cryptography, now featuring Samuel L. Jackson.

Some of the weaknesses in this algorithm can be fixed, as we illustrate below:

Algorithm. Vigenère ciphers. In a sense, the main weakness of the simple substitution cipher is that each plaintext letter always corresponded to the same encrypted symbol: this allowed us to use our knowledge of English to break the code. We can fix this, however, by using a Vigenère cipher!

Specifically: the key to a Vigenère cipher is a code word k that is some string of characters. To illustrate, suppose that the code word is "bah." To encrypt some message, like (for example) "Friendship is Magic!," we use the code word as a way to "shift" the letters of the codephrase as follows:

1. Write down your message. Below it, write a number of copies of the codeword so that each character in our message is matched up with a character from the codeword, as below:

```
Friendship is Magic!
bahbahbahb ah bahba
```

2. Then, take each character in the message, and "shift" it by the codeword character corresponding to it! In other words, take each codeword character, interpret it as a number (i.e. $a \mapsto 1, b \mapsto 2, \ldots$), and circularly shift the message character to the right that many places. For example, our message becomes

```
Hsqgoluiqr ja Obokd!
```

Example. We provide another example here, this one a bit more long-form. Consider the codeword "bode," and the message to be encoded[2]

```
The wasp and all his numerous family
I look upon as a major calamity.
He throws open his nest with prodigality,
But I distrust his waspitality.
```

In this setting, we would form the four lines

```
The wasp and all his numerous family
bod ebod ebo deb ode bodebode bodebo
I look upon as a major calamity.
d ebod ebod eb o debod ebodebod
He throws open his nest with prodigality,
eb odebod ebod ebo debo debo debodebodeb
But I distrust his waspitality.
ode b odebodeb ode bodebodebod
```

which results in the text

```
Vwi bcht fps eqn wmx pjqjtdyx hpqnnn
M qqdo zrdr fu p qfldv hcaerkic.
Mg ilwqlw trtr mkh rjui anvw twqsmlcamya,
Qyy K smxvgyxv wmx ypwukieqkic.
```

[2]**The Wasp**, a poem by Ogden Nash. He's great.

Weaknesses. This algorithm, given a sufficiently long codephrase, can avoid all of the issues that came up with our earlier work: given a codephrase that's like a few sentences long, then we would expect any given letter in our message to be shifted by many different values, and therefore that analyzing the character distributions will be useless.

And this is true! However, it is still weak to attacks that exploit the underlying structure of the English language. Specifically, there is a technique, called the **Kasiski test**, that we can use to break this code.

Roughly speaking, the Kasiski test is centered around the following observations:

- English has a lot of repeated sets of characters. We used this structure to great success in our earlier problem: we broke our earlier code largely by looking for repeated triples of characters and matching them up to known English characters.

- It is likely that our source message, being originally some large English text, has a number of often-repeated sequences. While it is possible that not all of those repeats will be preserved by the Vigenère cipher, (i.e. in our earlier text sample, the triple "him" occurred above the triple "ebo" the second time and "ode" the third time,) it is certainly likely that **some** triples will line up nicely with our code word, and therefore still occur as triples (for example, the first and second occurrences of "him" are matched with the same triple "ode.")

- Why do we care about these repeated phrases in our encrypted text? Well: if they correspond to repeated phrases in the original text (and aren't there just by accident,) then they tell us something about our codeword! In particular, they tell us that if our codeword sent those two repeated phrases to the same phrases, then both of those phrases were lined up over the "same" portion of our codeword!

- Therefore, there must be a whole number of copies of the codeword separating these two phrases, in order for them to line up! For example, using this observation on our text above with the repeated "his" phrase, if we count the number of letters between the first occurrence of "his" and the second occurrence of "his," we get 88. This tells us that it is likely that our codephrase is a divisor of 88; i.e. one of 2,4,8 or 11. Further analysis, by looking for more of these repeated sets, can eliminate other options, and tell us the precise length of the codeword we're studying.

- From there, we simply need to find the elements of the codeword! We can do this just like we did for the simple substitution cipher, basically. To be specific: break our cipher text into groups, each corresponding to the character of the code word that translated it. I.e. if we had the text from our earlier example and had deduced that the code word was length 4, we could then simply group our cipher text into four groups, each corresponding to the characters in the ciphertext that were shifted by a fixed codeletter.

  On each of these groups, we can then perform the character analysis we did before, and deduce one-by-one the codeword's characters.

5 Diffie-Hellman

All of the examples above have known and recognizable weaknesses. Consequently, they're not algorithms that we would want to use in a modern setting! Instead, let's consider the following algorithm, called the **Diffie-Hellman key exchange algorithm**: one of the first modern cryptographical systems!

Algorithm. The Diffie-Hellman key exchange algorithm works as follows: suppose that we have two people, Alice and Bob, that want to communicate over a public network.

Ahead of time, Alice and Bob agree on a public base prime p to work in, as well as a public seed g. Selecting this value p and seed g requires some thought in practice due to some complicated mathematical tricks people can perform. However, if you just pick a big prime p (like 300+ digits) this usually works pretty well. Choices for g are a little harder to pin down when they are "good," but usually most anything works — in practice, g is usually 2,3 or 5.

From there, Alice and Bob each pick a "secret key" a, b from $\mathbb{Z}/p\mathbb{Z}$. Then, they do the following:

1. Alice sends the number g^a publicly to Bob.
2. Bob then takes this received number g^a and uses his key to raise it to the b-th power, which gives him g^{ab}.
3. Bob then sends the number g^b publicly to Alice.
4. Alice then takes this received number g^b and uses her key to raise it to the a-th power, which gives her g^{ab}.
5. Bob and Alice are now both in possession of the same secret key g^{ab}. Any eavesdroppers will only have heard g^a and g^b, and thus have no obvious way to figure out g^{ab}.
6. To communicate a message $m \in \mathbb{Z}/p\mathbb{Z}$, then, either Alice or Bob can send to the other party $m \cdot g^{ab}$. No other parties know the secret key, so they cannot decode this message.
7. Moreover, because both Alice and Bob know the secret key g^{ab}, along with the values g^a, g^b, they can use Fermat's little theorem to calculate its inverse! Specifically: notice that Bob, knowing b, p, and g^a, can calculate

$$(g^a)^{p-1-b} = g^{a(p-1)-ab} = g^{a(p-1)} \cdot g^{-ab} = \left(g^{p-1}\right)^a \cdot g^{-ab}.$$

By Fermat's little theorem, we know that

$$g^{p-1} \equiv 1 \mod p,$$

and therefore that

$$\left(g^{p-1}\right)^a \cdot g^{-ab} \equiv g^{-ab} \mod p.$$

Therefore, Bob can calculate g^{-ab} and multiply it by their received encrypted message to get the original message back. Alice, using a, p, and g^b, can do the same trick as well; this allows both people to read their messages!

Why does this work? This is because the **discrete logarithm problem**, described below, is a "hard" problem to answer (in much the same vein as the NP problems we discussed earlier!)

Problem. The **discrete logarithm problem** is the following task: Suppose you're given a pair of numbers a, b in $\mathbb{Z} \mod p\mathbb{Z}$. The discrete logarithm problem asks the user for a value k such that

$$a^k \equiv b \mod p,$$

if some such value exists.

This problem is surprisingly hard to calculate. One naive algorithm that you might be tempted to try is simply raising a to higher and higher powers, and then reducing mod p each time until you got something that is equal to b. However, this takes a hideously long time to run: if we just take values of k starting at 1 and going up, we're effectively looking through the elements of $\mathbb{Z}/p\mathbb{Z}$ one at a time, in the order $a, a^2, a^3 \ldots$. If we assume that b is chosen more or less at random, then we'd expect to find b after going through about half of $\mathbb{Z}/p\mathbb{Z}$ (and in the worst-case after going through all of $\mathbb{Z}/p\mathbb{Z}$.) In either case, we have runtime that is linear in the size of the group! Therefore, if our group has say 300 digits (like we have for the size of prime requested in the algorithm above,) our algorithm has runtime that's absolutely awful (i.e. exponential in the number of digits in the size of the group.)

(It bears noting that this problem is an interesting one: it's both in NP, because it can be checked easily, but **unlike** many other problems it's not yet been shown to be NP-complete. So there's an odd chance that it might be in P after all; or at least it's more likely to be in P than things like the traveling salesman problem?)

As a consequence, Alice and Bob in the discussion above can be reasonably sure that (unless someone comes up with an algorithm that's much faster than the one above) their communications are going by un-eavesdropped, because everything they send needs the discrete log problem to be decoded without keys!