Cryptography Challenge

The Alan Turing Institute

2023-05-02

Table of contents

Overview

In this challenge, you and your team will learn about codes and ciphers: ways of scrambling a message such that they cannot be read by unintended people.

In the **first half of the day**, we'll be going through *symmetric ciphers*: these are ciphers where both you and the intended recipient share the same piece of knowledge (a *key*), which is used to both encode and decode the message. In particular, we'll spend time on *substitution ciphers*, where each letter of the alphabet is replaced with another according to some rule.

In the **second half**, we'll talk about *asymmetric ciphers*, where you and the recipient do not actually share a key. Although this might sound like a contradiction, this *is* possible, and is really important for the modern Internet, as you don't want to be sending your key over the Internet where anybody can read it!

1 Introduction to codes and ciphers

1.1 Codes and Ciphers

We may often talk of "code-breaking", or the "Enigma code", but in fact there is a subtle distinction between the meanings of *code* and *cipher*.

A *code* is a mapping from some meaningful concept (a word, or a sentence), to an arbitrary symbol (perhaps a letter or a number). For example, we might have a code that assigns the sentence "It's very cold today" to the number "67". There's no particular logic behind that, we just decided it, and wrote down this mapping in our *code book* so that it can be decoded later on.

Today though, we will be looking at *ciphers*. While a *code* operates on *meanings*, a *cipher* operates on *symbols* (such as individual letters). It transforms the "plaintext" symbols to their "ciphertext" counterparts using an *algorithm*. This algorithm will usually be a mathematical operation involving the original message and some sort of *key*. If someone knows (or is able to deduce) the algorithm and the key, they will be able to decipher an encrypted message.

1.2 Some basics

Suppose Alice wanted to send a message to her friend Bob, using a simple "mono-alphabetic cipher" where we replace each letter in our message with a different letter (we'll look in more detail at this type of cipher a bit later). The message might look like:

Tiuug Cgc,

Tgz oji lgd? Uiq'h riiq gs Rgseol!

Oupyi

> **Exercise:** Could we use some simple logic and guesswork to have a go at decrypting this? (When might Alice and Bob be planning to meet?)

Hints:

- What are common ways of greeting people?

- Since we know the names of both the sender and the recipient, could we look for those somewhere in the message?
- Look for things like double-letters, or places where the same letter appears in different words.

As a very basic step, even before we worry about what encryption algorithm to use, we can make life more difficult for someone who wants to snoop on our messages by taking a few simple steps:

- Only use capital letters.
- Ignore spaces, new lines, and punctuation.
- Put letters into e.g. groups of five.

It would be much harder for someone to try to decrypt:

TIUUG CGCTZ IGILG DUIQH RIIQG SRGSE OLOUP YI

than the message above!

Part I

Symmetric ciphers

As mentioned before, ciphers usually involve some sort of *key*, used to encrypt and decrypt messages. Ciphers where the *encryption key* and the *decryption key* are the same, are called *symmetric ciphers*. We will see a few examples here.

For symmetric ciphers, we need to worry about "key management". In order for someone to read the secret message you sent them, you first need to have shared the key - there's no point in having a sophisticated encryption algorithm if your adversary has they key! But how do you share it securely? (If you already have a secure communications channel for sending the key, why do you need to encrypt your message in the first place?)

In fact, many of the challenges in cryptography are related to designing *protocols* for ensuring that keys can be exchanged safely. The difficulty of sharing keys is also the motivation behind the development of *asymmetric* (or public/private key) ciphers, which we'll look at later on. For now though, we'll not worry about how we'd share keys in practice, and we'll look at some examples of symmetric ciphers.

2 Caesar cipher

2.1 Introduction

The Caesar cipher is the most basic type of encryption that we will look at. It is extraordinarily simple to use: each character of the alphabet is simply mapped to another character a fixed number of alphabet away.

For example, using a shift of 2, A would be turned into C, B into D, and so on, all the way until the end where X is turned into Z, Y back into A, and Z into B.

You can try out the Caesar cipher with this simple form here:

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

2.2 Breaking a Caesar cipher

Since there are only 25 useful possibilities for encoding, it is almost trivial to break a Caesar cipher. You can perform a **brute-force attack**, which means trying each possibility until you find one that gives the correct text. For example, use the playground above to figure out this hidden message:

```
OPKKLUTLZZHNL
```

3 Monoalphabetic ciphers

3.1 Introduction

Clearly, the Caesar cipher is not very secure. It's probably enough if you were sending an unimaginative message to a friend, but for anything mildly important, you probably want to crank up the security a little bit!

The Caesar cipher is a very simple example of a *monoalphabetic substitution cipher*: one where each alphabet is replaced with another one. This means that there's a *one-to-one mapping* between pairs of alphabet in the plain and cipher text. The problem with the Caesar cipher is that the replacement follows a very simple pattern, so once you know one mapping, the rest can be figured out right away.

We can go one step further and use a cipher where the mapping is completely random. For example, A can be mapped to B, but B can be mapped to Q, and C to I, and so on.

> **Question:** How many possible encryptions are there? Would a brute-force attack on such a cipher be sensible?

Try encoding a piece of text by choosing your own cipher. You can either enter each character of the cipher manually, or use the 'randomise' button to generate a random cipher.

A→

B→

C→

D→

E→

F→

G→

H→

I→

J→

K→

L→

M→

N→

O→

P→

Q→

R→

S→

T→

U→

V→

W→

X→

Y→

Z→

3.2 Frequency analysis

Although we realistically cannot use brute force, there is a much more clever way to crack such a code. It relies on the fact that certain letters of the alphabet are much more common than others in typical English text.

Try pasting some text into the box below (or clicking the samples), and observe the frequency distribution of the letters in the plot that appears:

Samples:

Question: Try a few different text sources. What are the most and least common letters? Can you think of any reasons why this distribution might systematically vary from text to text?

If you speak a foreign language, try analysing some text in that language to observe how the distribution might change. (Sadly, the box above ignores all accented characters! The schemes we're discussing today can be adapted to work on non-English letters, but today we'll focus only on the 26 English alphabet.)

In English, the most common letter is by far 'E'. If we perform the same analysis on the cipher text, and find that 'R' is the most common letter, then it's likely that 'R' decodes to 'E' in the plain text.

Once we have a match, we can fill it in and try to solve the rest in an iterative manner.

Another useful piece of information you can get from the cipher text is to find repeated sequences of letters. For example, once you find the letter for 'E', it makes sense to look for potential spots where 'THE' might be encoded.

3.3 Decryption

With the above information, you should be able to have a go at decrypting this encrypted text, with the frequency distribution shown here:

A→

B→

C→

D→

E→

F→

G→

H→

I→

J→

K→

L→

M→

N→

O→

P→

Q→

R→

S→

T→

U→

V→

W→

X→

Y→

Z→

4 Vigenère cipher

4.1 Introduction

Both the *Caesar cipher* and the *Monoalphabetic substitution cipher* have a single, fixed mapping from input letters to ciphertext letters, making them vulnerable to letter frequency analysis. It would be more secure if we could create a *polyalphabetic cipher* - i.e. the mapping from plaintext letters to ciphertext letters changes after every input character. One example of this is an extension of the Caesar cipher, called a "Vigenère cipher" (named after the 16th-century French diplomat Blaise de Vigenère).

In this cipher, we define the key as a word or phrase that is repeated as many times as necessary to cover the length of the message. Each letter in this key is then used as the key for a Caesar cipher to encrypt the corresponding letter in the message.

For example, if the message is "Cryptography is fun" and the key is "SECRETKEY", we would write the message and the repeated key together:

```
CRYPTOGRAPHYISFUN
SECRETKEYSECRETKE
```

Since the first letter of the key is "S", which is the 19th letter of the alphabet, we shift the first letter of our message ("C") by 18 (since we start counting with "A" as zero, meaning "no shift"), to give us "U".

> **Exercise:** Using pen and paper, go ahead and encrypt the rest of that message.

UVAGXHQVYHLAZWYER

4.2 Cracking the Vigenère cipher

Luckily, the Vigenère cipher still contains a point of weakness that we can use to attack it: the *periodicity* of the key, or in other words, the fact that it repeats itself every n characters.

This means that, if the key length is 3, then the 1st, 4th, 7th, … characters are encoded in the same way, and we can tackle it just like we did for the simple ciphers seen already.

But, to do this, we first need to know the key length!

4.2.1 Index of Coincidence

To find the key length, we need to understand another fundamental property of English text: the *index of coincidence* (IoC). If we have a piece of text and randomly pull two letters out of it, the IoC tells us how likely it is that these two letters are the same.

Mathematically, we can define the IoC as:

$$\text{IoC} = 26 \cdot \frac{n_\text{A}(n_\text{A}-1) + n_\text{B}(n_\text{B}-1) + \cdots + n_\text{Z}(n_\text{Z}-1)}{N(N-1)},$$

where n_i is the number of times the letter i appears in the text, and N is the length of the text.

> **Question:** Show that if all letters have the same probability of occurring, then the IoC is equal to 1. (Assume that the text is long enough, such that the n_i's are much larger than 1.)

Because letters are not uniformly distributed in English (recall that 'E's were particularly common), this quantity is, on average, greater than 1 for typical English texts.

Have a play around with this plot, and find out what the IoC is for some typical text. The same samples as before are included, plus a random sequence of letters:

Samples:

4.2.2 Using IoC to determine Key Length

We can exploit the fact that the IoC has different values for English text and for random sequences of letters, to identify the key length. If the key had a length of e.g. **4**, we would know that the 1st, 5th, 9th, … letters were encrypted with the same letter of the key, and we would expect the IoC of this collection of letters (and also the 2nd, 6th, 10th, …) to be higher than what we would see for random. If on the other hand, the key had length **5**, we would expect the same to be true if we looked at the 1st, 6th, 11th, 16th, … letters. (This periodic property is going to be looked at in more detail in "Modular Arithmetic", which we'll cover this afternoon.)

> **Exercise:** Decrypt the following ciphertext.
>
> **Part 1:** First, look at the Index of Coincidence and Letter Frequency graphs for the subsets of the ciphertext that we'd expect to see if the key length were 2, 3, or 4.

Choose a key length to try:

Proposed key length:

> **Part 2:** Now we have identified the most likely key length, we should be able to look at the individual letter frequency graphs, and work out how much we'd need to shift them (like a Caesar cipher) to identify each letter of the key.
>
> **Hint 1:** Remember, the most common letter in English text is "E".
>
> **Hint 2:** When we encrypt a letter using the Vigenère cipher, the letter "A" corresponds to a shift of 0. So, if the most common letter in the first of the graphs above was "F", this would imply that we had a Caeser cipher of shift=1 (to move "E"s to "F"s), corresponding to the first letter of the key being "B".

Proposed Key:

4.3 Aside: One-time pads - the unbreakable cipher

Since the way that we break the Vigenère cipher depends on the repetition of the key, it follows that if the key is (at least) as long as the message, we wouldn't be able to crack the cipher this way. In fact, these so-called "one-time pads", where the key is a long sequence of randomly generated letters, are thought to be unbreakable (as long as the sequence is truly random, and is not re-used - hence the name). These were used for some of the most secret communications in World War II. However, of course, we still have the problem of how to securely share the key between the sender and the recipient!

5 Enigma

5.1 The Enigma cipher machine.

The *Enigma machine* was used by the German military during World War II to decode and encode secret communications. The simplest version of the machine consists of:

- A typewriter-style keyboard, which could be used to type out the message.
- A set of lamps, one for each letter of the alphabet. One of these lamps will light up every time a key on the keyboard is pressed.
- Three (or later four) rotors, chosen from a selection of five, each of which has 26 electrical contact pins on each side, and a different mapping of connections between them.
- An electrical *reflector* next to the left-hand rotor.

The choice of which rotors to use, in which slots, and their starting rotations, constitutes the encryption (and decryption) *key*.

When the machine is setup in the specified starting position, the user can press a key on the keyboard, and an electrical current will flow through the rotors from right-to-left, then through the reflector, then back through the rotors from left-to-right, and will cause a letter-lamp to light up. At least one of the rotors will then rotate by one step (the right-hand rotor rotates after every key press, the others less frequently), so that when the next key is pressed, the mappings from the keyboard to the lamps will be different.

Like Vigenère, Enigma is therefore a *polyalphabetic substitution cipher*: a given configuration of the machine is a mono-alphabetic substitution cipher, but we get a different configuration after every key press.

Since Enigma is a symmetric cipher, the procedure is exactly the same for encrypting a plaintext message, or for decrypting a ciphertext message.

> **Exercise:** Have a go with the Enigma emulator. Some things to try: - Set the starting rotors to something (e.g. "A,B,C"), then type out a short message. Note that after every keypress, the right hand rotor shifts by 1. - After the end of that message, reset the rotors to the original position, and then type out the ciphertext that you got from the previous step. Hopefully you should now recover the original text. - You can also try pressing the same key many times - what do you notice about the output ciphertext?

5.2 Cracking the Enigma

The *reflector* next to the rotors gives the Enigma the nice property that the settings for encryption and decryption are identical. However, it also gives rise to an important flaw (which you may have noticed from the exercise above): *it is impossible for a letter to encrypt to itself.* This means that if you have a "crib" (i.e. a word or phrase that you are fairly sure will appear in the message), and you are trying out lots of possible keys, you can discard any where the same letter appears in the same position in the ciphertext and the proposed plaintext.

Using this, and various flaws in the procedures that Enigma operators used, Polish cryptographers were able to break Enigma in the 1930s. They then shared this knowledge with British and French cryptographers, and a cryptographic arms race ensued that lasted throughout World War II. As extra layers of complexity were added to Enigma (a fourth rotor, a plugboard that mapped pairs of letters to one another, …), so the scale and complexity of the code-breaking efforts increased. An example of this is the development of the *Bombe* - an electro-mechanical device that could try all the 17576 (26x26x26) possible rotor positions in about 20 minutes.

Figure 5.1: The bombe at Bletchley Park. Image by Antoine Taveneaux (license: CC-BY-SA 3.0)

Part II

Asymmetric ciphers

Asymmetric cryptography refers to an encoding/decoding algorithm that requires two separate keys, known as a *public key* and a *private key*.

Let's say Bob wants to send Alice an encoded message that only she can read. Alice must first generate both a public key and a private key for herself. She sends the public key to Bob, and keeps the private key to herself.

Bob can then use the public key to encrypt his message to Alice; and Alice uses her private key to decrypt it. Since only Alice has the private key, only she can decrypt the message.

You can think of this as Alice giving Bob a lock, but keeping the key for herself.

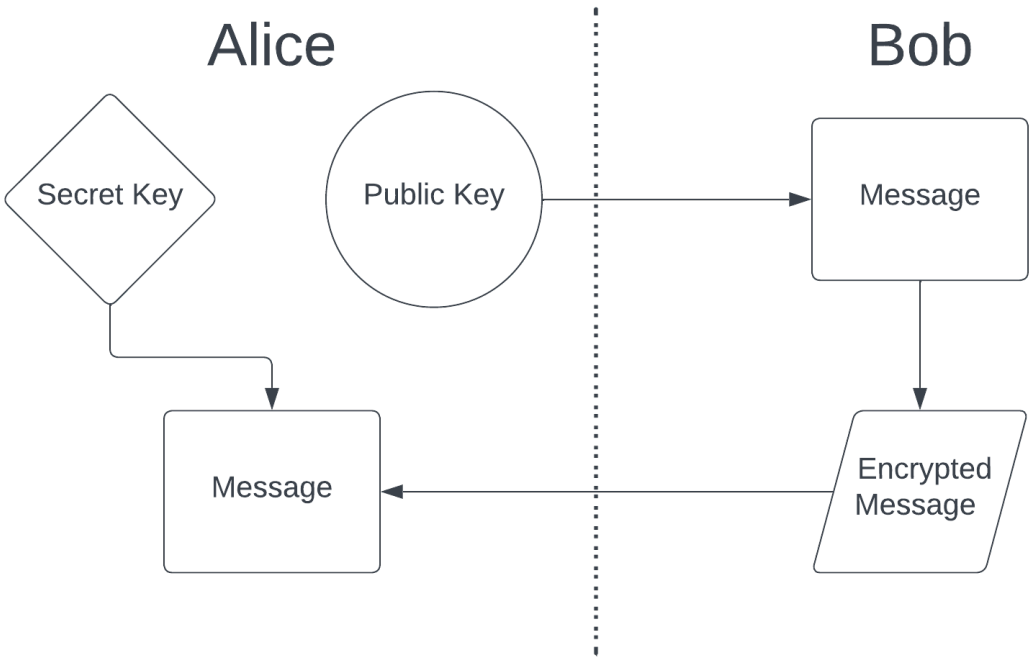

Figure 5.2: Alice gives Bob her public key so Bob can encrypt a message, which she then decrypts with her private key.

In this section, we'll look at one specific example of an asymmetric scheme, namely the RSA system.

6 Modular arithmetic I

To begin the second part of this challenge, we'll take a brief look at *modular arithmetic*. This is the same thing as regular arithmetic, but with one twist: we only care about the remainder when dividing by some number n.

Instead of talking about numbers, we talk about numbers *modulo* n. For example, $5 \bmod 3 = 2$, because the remainder when dividing 5 by 3 is 2. But also, $8 \bmod 3 = 2$; so 5 and 8 are equivalent, or *congruent*, modulo 3.

Essentially, a and b are congruent modulo n if a and b have the same remainder when divided by n.

This is often denoted as:

$$5 \equiv 8 \bmod 3.$$

We can insert any other expressions we like in here, as well:

$$1 + 4 \equiv 4 \times 2 \bmod 3.$$

Another way of thinking about this is that integers 'wrap around' back to 0 after reaching $n - 1$. So, when counting *modulo* 3, instead of

$$0, 1, 2, 3, 4, 5, 6, 7, 8, ...$$

we have:

$$0, 1, 2, 0, 1, 2, 0, 1, 2, ...$$

Because 5 and 8 both 'become' the same number, they are congruent modulo 3.

> **Question:** Consider some other areas in life where we use modular arithmetic without thinking about it.
>
> What day of the week will it be, five days from now?
>
> If you need to wake up at 7 am and you need 8 hours' sleep, what time do you need to sleep?
>
> Can you express these relations using modular arithmetic?

6.1 Calculator

To help you get a feel for modular arithmetic, here is a calculator that accepts any mathematical expression in the left-hand side. You can also specify a modulus, and the calculator will show you the result of the expression modulo that number. You may find this useful for the exercises below.

mod

Answer:

6.2 Exercises

Suppose that $a \equiv p \bmod n$ and $b \equiv q \bmod n$. Prove the following statements:

1. $(a + b) \equiv (p + q) \bmod n$.
2. $(ab) \equiv (pq) \bmod n$.
3. $(a^m) \equiv (p^m) \bmod n$, for all non-negative integers m.

These equalities make it far easier to perform some calculations. For example, what is $33^{594} \bmod 32$?

We might first think about calculating 33 to the power of 594, and *then* dividing that by 32 to find the remainder. That sounds difficult: even a computer would struggle with such a large number! (Try using the calculator above, or Google's built-in calculator, to calculate 33^{594}.)

But if we notice that $33 \equiv 1 \bmod 32$, then we can use formula (3) to see that

$$33^{594} \equiv 1^{594} \equiv 1 \bmod 32,$$

because 1 to the power of anything is still 1.

Challenge: Calculate $7^{175} \bmod 16$.

7 RSA scheme I

The RSA scheme is one of the most important asymmetric-key systems, and is widely used as part of the steps in encrypting Internet communications.

The RSA scheme is named after the surnames of its three inventors: Ron Rivest, Adi Shamir, and Leonard Adleman. Although they were the first to publicly describe the RSA system in 1977, an equivalent algorithm had in fact been discovered four years ago by Clifford Cocks, who worked at GCHQ.

7.1 Introduction

The RSA scheme revolves around modular arithmetic, which the previous page touched briefly on.

Specifically, it assumes that the message you want to transmit is first translated into an integer m (which stands for message).

> **Question:** Can you come up with a way of translating a plain-text message into an integer?

1. The *encoding* step involves the sender taking the message m and raising it to a power e (for 'encryption'), modulo some number n:

$$c \equiv m^e \mod n$$

   where c is the *ciphertext*. The combination of e and n forms the *public key* of the RSA scheme.

2. This ciphertext is then passed to the recipient, who then *decodes* it by raising it to a power d (for 'decryption'), modulo n:

$$m \equiv c^d \mod n.$$

   d forms part of the *private key*, which only the recipient is allowed to know.

The two equations above imply together that:

$$m \equiv (m^e)^d \mod n,$$

and crucially, this is true of *all* messages $m < n$, meaning that the keys can be used for whatever message you choose to send.

In general, if we just pick any random numbers e, d, and n, this will not be true! The RSA scheme works because these numbers are specifically chosen in a way that not only satisfies the equation above, but is also difficult to reverse-engineer.

Specifically, if you were an attacker and you knew the public key (i.e. e and n) *as well as* the ciphertext c, it's still extremely difficult to find the correct value of d to correctly decode the message.

7.2 Key generation

Because the RSA scheme is asymmetric, it only works for communication in one way. The *recipient* is responsible for generating the three integers e, d, and n. They do so using the following algorithm:

1. Choose two prime numbers p and q, and set $n = pq$.

   For example, if we choose $p = 5$ and $q = 13$, then $n = 65$.

2. Calculate the *totient function*, defined by $\phi = (p-1)(q-1)$. (That symbol is the Greek letter phi.) In this case, we would have $\phi = 4 \times 12 = 48$.

   Then, choose an integer e which is smaller than ϕ and has no common prime factors with ϕ. Here, the only prime factors of 48 are 2 and 3: so, a valid choice would be $e = 5$.

   e and n are part of the public key, and can be shared with the sender.

3. Finally, choose d such that $de \equiv 1 \mod \phi$.

   In this case, we need $d \times 5 \equiv 1 \mod 48$: the smallest value of d for which this holds true is $d = 29$. (Can you verify this?)

   d is part of the private key, and must be kept secret.

This setup ensures that regardless of what message m is being sent,

$$(m^e)^d \equiv m \mod n,$$

as required by the RSA algorithm.

Try this out using the interactive form here. It has been pre-filled with the numbers from the example above, but you should try it out with your own choice of numbers.

- **Step 1**: Choose p and q (must be prime)
  p: q:
- **Step 2**: Choose e (encryption key: must be smaller than ϕ, and cannot share prime factors with ϕ)
  e:
- **Step 3**: Choose d (decryption key: must satisfy $de \equiv 1 \bmod \phi$)
  d:
- **Step 4**: Choose m (message: must be smaller than n)
  m:

7.3 Next steps

We've managed to show here that the RSA scheme *works*. However, there are several questions that remain:

1. Unless you chose very small numbers for p, q, and e, you probably found it hard to calculate d. How can this be done quickly?
2. How do we know that the RSA scheme is *secure*? Is it possible to obtain the private key, d, if you have the public key (e and n)?
3. why does the RSA algorithm work at all, i.e. why is it always true that $(m^e)^d \equiv m \bmod n$?

We'll cover the first two questions in the remainder of today.

The proof that RSA works is provided on a bonus page, although it doesn't form part of the activities for today, as it requires slightly more mathematical knowledge than we've seen so far.

8 Modular arithmetic II

8.1 Introduction

In this section, we'll look at the *extended Euclidean algorithm*. An algorithm is a series of steps which can be used to solve a problem: think of it as a recipe, but for maths.

In this case, the problem we're trying to solve is finding the values of x and y in the equation

$$ax + by = \gcd(a, b) \tag{1}$$

where a and b are known integers, x and y are unknown integers, and $\gcd(a, b)$ denotes the *greatest common divisor* of a and b, i.e. the largest whole number that divides both a and b.

Formulated this way, this equation might not immediately seem relevant. However, in fact, it is precisely what we need to find the value of d in the RSA algorithm.

Recall that we chose e and d such that:

- e and ϕ have no common factors. This means that $\gcd(e, \phi) = 1$.
- $ed \bmod \phi = 1$.

  Question: Reformulate the second condition in terms of equation (1).

8.2 The extended Euclidean algorithm

You probably found above that we are trying to solve the equation $ed + \phi y = 1$. We already know the values of e and ϕ, and the Euclidean algorithm will let us obtain the values of d and y.

Of course, we're only really interested in d, so even though we'll also get the value of y, we can just throw it away.

Let's reuse the example from the previous page, where $\phi = 48$ and $e = 5$. Substituting that in:

$$5d + 48y = 1$$