About Project

**Information security**

[FIG: Stock image related to data]

The modern world is full of sensitive data. Servers on the Internet hold our credit card information, medical records, embarrassing music playlists, and less-than-flattering selfies. It’s hard to keep all that data safe from digital snoops. Almost every day there’s a new story about hackers stealing secrets. The field of <i>information security</i> has never been more important, but few people understand it. Even if you don’t write code or handle data for a living, it’s important to understand how your information can be protected – and how bad guys can break those protections!

**Send secret messages on the BBC micro:bit**

In this project, you’ll use your BBC micro:bit, a tiny computer board that anyone can program, to send and receive secret messages. Program a text message or a small image into the micro:bit, then transmit it over the wireless radio to another micro:bit. With no protection, anyone else with a micro:bit could read the message too. You’ll learn how to scramble the message so that it can only be read by people you want to see it.

The tutorial [link] will teach you several different ways to protect your messages. You’ll learn how to use the Caesar cipher, which was used to send military strategies in ancient Rome… and then learn how to crack it! We’ll also show you more modern methods that are used for data on the Internet. Try designing your own secret code and use it to send messages to your friends.

**Teaching tools**

[TODO?]

**Going further**

Once you finish the tutorial, you can download our code library [link] to try more advanced encryption schemes. You can even combine different encryption methods to make your messages even more secure.

This project just scratches the surface of information security. To learn more, check out some of these resources:

[Links]

Want to learn more about programming with the BBC micro:bit? Check out some of these other cool projects:

**About us**

[FIG: team photo]

This project was developed for the ARM Intern Innovation Challenge, a competition among ARM summer interns to design fun and educational projects with the BBC micro:bit. Here’s our team:

[Team info]

Built It!

**Getting started with the BBC micro:bit**

[FIG: picture of micro:bit]

The micro:bit is a tiny computer that anyone can program. The board comes with lights, buttons, and a wireless radio, which we’ll use to send messages from one micro:bit to another. If you haven’t used the micro:bit before, check out the Getting Started [link] guide from the micro:bit website.

You’ll need two micro:bit boards, one to send a message and one to receive it. We’ll be using the JavaScript development tools [link] in this tutorial. You can also download code [link] for micropython.

**Sending messages over the radio**

To start with, let’s send a message over the radio. Wireless communication is pretty complicated: the text has to be converted into a number, then into digital symbols, then into different radio waves, and finally sent through circuits to an antenna. Fortunately, the micro:bit does all the hard stuff for you!

On the sending micro:bit, use this code:

[Send code]

The first line tells the radio what channel to use. Both micro:bits need to be on the same channel in order to talk to each other. The radio.sendString function tells the micro:bit to transmit the word “Taco” from the radio. [Describe function for button press action]

On the receiving micro:bit, use this code:

[Receive code]

The first line is the same as in the send code. It makes sure both micro:bits are on the same channel. The rest of the code tells the micro:bit what to do when it receives a message from the radio. In this case, it displays the message on the screen.

Try it with your two micro:bits. The receiving micro:bit should show the word “Taco” on the screen when you push the button on the sending micro:bit.

[FIG: photo of working message transmission]

**Keeping messages secret**

[FIG: eavesdropping cartoon]

When the micro:bit sends a message over the radio, any other nearby micro:bit on the same channel can see it. The Internet works the same way: when you send a “packet” of data to the server, everyone else on the same network can see it too. How can we protect the information when anyone can see it?

[FIG: scrambling messages]

To protect a message that’s transmitted in public, we have to scramble it so that the person who’s supposed to see it can unscramble it, but no one else can. In the world of information security, the scrambled message is called <i>ciphertext</i> and the rule that’s used to scramble it is called a <i>cipher</i>. For a cipher to work, it has to combine the message with another secret that only the sender and receiver know. This secret is sometimes called a <i>key</i>. Anyone who knows the key can decipher the message, and everyone else will just get nonsense.

[Inset] It’s important to keep the key a secret, or else anyone can decipher the message. The strongest lock in the world is no good if everyone has the key! The same is true of passwords. The easiest way to steal secret information from an encrypted system isn’t to break the code – it’s to trick someone into telling you their password.

**The Caesar cipher**

The first cipher we’ll program is called the <i>Caesar cipher</i>. It’s named after the ancient Roman ruler, who reportedly used it to send orders to his armies. The Caesar cipher is a kind of <i>substitution ciphers</i>, in which each letter is replaced by a specific other letter. Here’s an example:

[FIG: Caesar cipher + 1]

Can you guess how it works? Every letter is replaced by the next letter in the alphabet. To make it harder to predict, we can use other offsets. For example, with an offset of 4, we’d replace A with E, B with F, etc. When we get to Z, we have to wrap around and start with A again. So with the offset of 4, W would be replaced with A, X with B, Y with C, and Z with D.

To program the cipher into a computer, we first convert every letter into a number: A is 0, B is 1, and so on. Then we add the offset. To wrap around, we use the “modulo” operation, which is the % sign in computer code. Then convert the number back into a letter and send it over the channel. Here’s the code:

[Caesar transmit code]

The receiving micro:bit does the same thing, but it subtracts the offset instead of adding it:

[Caesar receive code]

Try transmitting a message again. If the receiver knows the offset that the sender used, it should display the same message that was sent. Now try changing the offset and see what happens. The cipher makes it much more difficult to eavesdrop on the radio transmissions.

**Building better ciphers**

Caesar ciphers are easy to understand and code, but they’re also easy to break. You just need to figure out the offset. It wouldn’t take very long to try all 26 possible offsets. One way to improve the cipher is to assign the substitute letters in an arbitrary way: A to M, N to L, D to X, and so on. Just make sure that every letter has a different substitute. Then there are about 403 million billion billion ways to assign the letters!

[Inset] Since there are so many possible combinations of letters, it’s much harder to break this cipher by brute force, but it can be done by looking for patterns in the letters. Some letters occur more often than some other letters – E is much more common than X, for example – and some letters often appear together, like Q and U. Computers are very good at finding these patterns, so substitution ciphers are not very secure today.

Here’s a different kind of cipher that also uses the “modulo” function. Can you figure out how it works? Can you figure out how to break it?

[Code for addition cipher]

**Protecting other kinds of data**

So far, we’ve scrambled text data by replacing some letters with others. But what if we want to protect an image, or an audio clip, or any other type of digital file? Computers store and transmit information as 0’s and 1’s. These numbers are called “bits”, and they can be used to represent any kind of information in a computer. If we can design a way to scramble the 0’s and 1’s, we can protect any type of data. The encryption schemes used on computers today all treat data as numbers, no matter what kind of information the data represents.

The micro:bit makes it easy to view and send this kind of data. We’ll use the display to show an “image” made up of 25 lights. A light that’s on represents a 1, and a light that’s off represents a 0. There are 25 lights on the micro-bit, so our message will have 25 bits. First, let’s write some code to send and receive these images:

[Send code]

[Receive code]

[Note: should separate the part of the code that displays the image from the part that encodes and decodes it]

**Bit flip cipher**

Now let’s protect this image. We’re going to use a bit flip cipher, uses another “image” of 0’s and 1’s as its secret key. To encode the message, line up its bits with the bits of the secret key. If the secret key has a 1, then flip the bit of the message. If the secret key has a 0, then leave that bit of the message alone.

[FIG: bit flipping diagram]

If anyone tries to look at the image without knowing the secret key, they’ll just see random bits!

Here’s the code for the sender:

[Transmitter code]

And here’s the code for the receiver:

[Receiver code]

Do you notice anything strange about the code? The encoding and decoding operations are exactly the same! Every bit that has been flipped gets flipped back to where it started, and the bits that weren’t flipped are left alone.

Bit flipping ciphers might seem strange, but they’re incredibly useful. Not only are they simple, [b]they’re completely unbreakable[/b]! If the cipher is used correctly, it is mathematically impossible to learn any information about the message. What’s the catch? To guarantee that it’s unbreakable, each set of secret bits can only be used once. The more messages that are sent with the same key, the less secure the messages become.

[Inset] There’s one other catch to bit flip ciphers. To be secure, the pattern has to be totally random. The “random” numbers generated by computer software are good enough for shuffling music or rolling dice in games, but they’re not random enough for encryption. Some computer chips have special features to generate better random numbers using noise from electronic circuits. Randomness lies at the heart of nearly every encryption scheme!

Because of this one-and-done rule, the bit flip cipher is known as a <i>one-time pad</i>. Since it’s difficult to generate and share a different key for every message, these ciphers are only used for the most sensitive messages.

[Inset] The key is often the weakest link in an encryption system. A lot of cutting-edge security research today is on secure ways to create and share keys over networks. Some scientists are developing ways to use quantum physics to share keys over special fiber-optic cables. The safest way, of course, is to hand the key to someone in person.

**Modern Encryption**

The ciphers in this demo project can be used to protect messages from eavesdroppers. They’re much better than sending a plain message, but they each have weaknesses. The substitution ciphers can be broken by a determined adversary. The one-time-pad, while secure, isn’t practical for everyday use.

Today’s state-of-the-art encryption systems use a combination of these approaches. The data is encoded using a bit flip cipher, but it’s not a randomly generated one-time-pad. Instead, the bits are created according to a rule from a smaller secret key. These methods are breakable in theory, but to decode the messages without the secret key, an adversary would have to solve a math problem so hard that even our fastest computers would take centuries to do it. These methods are also much too complicated to put onto a micro:bit.