

## How it works

- Passwords are security measures to prevent unauthorized access to accounts, locks, and devices.
- A password may be stored on a computer or device in a way that is not readable by people. If hackers break into your computer, they won't find your **cleartext** password. Instead, they see an encrypted version of the password called a **hash**. When an account is created, the cleartext password is scrambled by a mathematical **hashing function** that cannot be reversed, such as **SHA-256**. The output of a hashing function is the hash and is 256 bits in length. The hash is stored on the micro:bit as a 64-character **string** in a file named "password.txt".
- During login, an **authentication** process is used to verify if a user's password is correct. During authentication, the hash stored in the "password.txt" file is read from the micro:bit and compared to the hash of the password just entered. If the two are equal, the user is granted access.
- A hacker can't use a password's hash like the cleartext password because, during authentication, the hash would be re-hashed and fail authentication.

## What will you do?

1. Practice hashing:
  - a. Open "practice\_hash.py" in the editor and run the program. Enter a password at the prompt to display the hash string. Note – The string is 64 characters or 256 bits.
2. Set password to micro:bit:
  - a. Open "set\_password.py" in the editor and run the program. Enter a password and the hash will be saved to the file named "password.txt" on your micro:bit. Note – the hash is a 64-character string; each character requires one byte of memory when stored on the micro:bit.
3. Read hash from micro:bit:
  - a. Open "read\_password\_hash.py" in the editor and run the program. The hash stored in the file named "password.txt" during 2a will be read from the micro:bit and displayed. Note – the password string is not the cleartext password.
4. Practice authentication:
  - a. Open "authentication.py" in the editor and run the program to test your password and the authentication routine. This program compares the hash stored on the micro:bit to the hash of the entered password. If the two are equal, the user is granted access.
5. Remote login:
  - Ensure all group members use the same assigned group number.
  - The **receiver**:
    - a. *Whisper the password of YOUR micro:bit to the sender.* Open 'student\_receiver.py' in the editor and change the group to your assigned number, and then run the program *before* the sender has run theirs.
  - The **sender**:
    - a. Open 'student\_sender.py' in the editor and change the group to your assigned number, and run the program *after* the receiver and hacker have started theirs. Enter the *password of the receiver's micro:bit*. You will be attempting to remotely log in on the receiver's micro:bit.
  - The **hacker**:
    - a. Open 'student\_hacker.py' in the editor and change the group to your assigned number, and then run the program *before* the sender has run theirs. You will be doing a **man-in-the-middle attack**.

- Discuss among the group what message was sent over the radio. Can the hacker use the message they received to authenticate the receiver's micro:bit? Change roles and try again.

### Code it

#### Sender role

```
EDITOR: SEND_4
PROGRAM LINE 0001
from microbit import *
from mb_radio import *
from hashing import *
from mb_disp import *
from mb_music import *

radio.on()
radio.config(length=250, channel
            =12,power=6,group=1)
disp_clr()
pswd = input("Enter password: ")
```

#### Receiver role

```
EDITOR: RECV_4
PROGRAM LINE 0001
from microbit import *
from mb_radio import *
from mb_file import *
from mb_disp import *
from mb_music import *

radio.on()
radio.config(length=250, channel
            =12,power=6,group=1)
disp_clr()
print("Waiting for message...")
```

#### Hacker role

```
EDITOR: HACK_4
PROGRAM LINE 0001
from microbit import *
from mb_radio import *
from mb_file import *
stolen_hash=""
radio.on()
radio.config(length=250, channel
            =12,power=6,group=1)
disp_clr()
print("Man-in-the-middle attacki
ng...")
while not escape():
```

### Go further

- Run the “authentication.py” program again and enter the wrong password three times in a row.
- Try remote login again, but send the hash message the hacker received instead of the cleartext password.
- Try a different role in your team.

### Check your understanding

- The password hash, not the cleartext password, is saved on the micro:bit in the file named “password.txt.”
- A hacker can’t use the hash as the password during login; it would be re-hashed and fail authentication.
- A hashing function is a mathematical encryption. It is one-way; in other words, it cannot be reversed.
- A unique password has one, and only one, hash.

### Help

- Ensure the receiver and hacker run their program before and wait for the sender to transmit the message.
- Ensure all group members use the same assigned group number.
- Ensure passwords are successfully set on each micro:bit.

### Files

- Transfer the activity files below to your calculator using the TI Connect CE Software. The link to download is [here](#). The best practice is to load all files for this cybersecurity activity and then delete them before loading the next set of activity files. This helps keep your calculator organized.

| Name        | Description   |
|-------------|---|
| pract_4.py  | Displays the hash of the password and displays the length bytes and and bits. |
| set_pw_4.py | Prompts for password and saves the hash in ‘password.txt’ file on micro:bit.  |
| rd_pwd_4.py | Reads and displays the hash stored in ‘password.txt’ on the micro:bit.        |
| authen_4.py | Compares the hash of entered password with the hash stored on micro:bit.      |
| send_4.py   | Sends hashed password to receiver for authentication.                         |
| recv_4.py   | Receives hashed password and authenticates.                                   |
| hack_4.py   | Man-in-the-middle attack to snatch the hash from the sender.                  |
| HASHING.8XV | SHA-256 hashing module  |