

How it works

- A cryptographic hash function is one-way; the original password can't be calculated from its hash. When a hacker breaks into a computer, the password hashes could be stolen but can't be used to log in. A **brute-force** attack (activity 6), which tests many cleartext passwords, can be prolonged and requires a lot of computer power and time. A faster method is to search an existing table of passwords and their **pre-computed** hashes. A **Rainbow table** is a Python **dictionary** of pre-computed hashes of frequently used passwords. If a stolen hash matches one in the table, the corresponding cleartext password can be retrieved from the table and used by the hacker to log in to that computer.
- Hackers may work together on multiple distributed computers to pre-compute huge rainbow tables of commonly known cleartext passwords, such as 'abc123'. Rainbow tables are clandestinely shared on darknet servers for download and use by bad actors who may attempt to break into an account using a common password.
- Using a common password for a secure account leaves the user vulnerable to a rainbow table attack.
- Searching a rainbow table is very fast, but the tables can be huge.
- Rainbow tables are frequently updated with new passwords.
- Long passwords with infrequently used characters are less likely to appear in a rainbow table.
- Changing your password frequently and practicing good **password hygiene** helps to reduce your chances of being a victim of a rainbow table attack.
- A **salt** is a keyword added to all passwords in the set password and authentication program; it is set by the cybersecurity programmer on a particular **platform**. The salt added to the password differs for each platform, such as a web banking or shopping account.
- If a person uses the same password across several platforms, the stored hash will be different because the unique salt added to the cleartext password is different on each platform. This technique helps to secure the accounts of individuals who use the same password on multiple accounts. The cybersecurity programmers set the salt and are unique to each platform.

What will you do?

1. Practice using a rainbow table:

Ten commonly used passwords

<i>Password</i>	<i>qwerty</i>	<i>111111</i>	<i>abc123</i>	<i>12345678</i>
<i>123456</i>	<i>guest</i>	<i>123123</i>	<i>123456789</i>	<i>12345</i>

- a. The rainbow table 'rbt' contains the hashes of these ten passwords.
- b. Open 'prac_1_5.py' in the editor and run the program. The hash of one of the passwords in the table above will be randomly selected and saved to the file 'practice_password.txt' on your micro:bit. Look at the displayed hash. Can you read it and know which password was selected?
- c. Open 'prac_2_5.py' in the editor. Notice that 'rbt_5.py', the rainbow table, is imported into this program. Run the program. The password hash saved in 1.a will be read from the micro:bit and stored in the variable named "hash" in the Python shell. Press the [var] key and select the variable name "hash" from the menu to display the hash in the **Python shell**.
- d. Use the rainbow table named 'rbt' to look up the password corresponding to the hash. To use the dictionary table in the Python shell, type `>>>rbt[hash]` at the **REPL prompt**. Hint: Use the [var] key to paste "hash" into the index of the rbt[] dictionary. Which password was selected in 1.b?
- e. Repeat steps 1.b and 1.d to practice using the rainbow table a few more times.

2. Set a password and authenticate on the micro:bit.
 - a. For this activity, choose one of the ten commonly used passwords in the table above.
 - b. Open “set_pw_5.py” and run the program to set a new password on your micro:bit.
 - c. Open “pract_3_5” and run the program to display the hash of your micro:bit password. Write down the first five characters of the hash on a piece of paper to compare with the salted hash in activity 4.c.
 - d. Open “authentication.py” and run the program to test your new password.

3. Remote login:
 - a. Ensure all group members use the same assigned group number.
 - The **receiver**
 - a. *Whisper your password to the sender* so the sender can log in to your micro:bit. Be sure to *keep the password private* from the hacker. Advance to ‘student_receiver.py,’ change the group to their assigned number, and run the program **before** the sender has run theirs.
 - The **sender**
 - a. Advance to ‘student_sender.py,’ change the group to your assigned number and run your program **after** the receiver and hacker have started theirs. *Send the receiver’s password* to log in to their micro:bit remotely.
 - The **hacker**
 - a. advance to the ‘student_hacker.py,’ change the group to their assigned number, and then run the program *before* the sender has run theirs. Note – this program *will receive the password hash* sent by the sender to log in to the receiver’s micro:bit.
 - b. Use the stolen hash and the rainbow table, as you did in step 1.c, to look up the receiver’s password from the intercepted hash.
 - c. Once the hacker has cracked the receiver’s password, the receiver should run the ‘student_receiver.py’ program again to test if the hacker can break into your micro:bit.
 - d. The hacker should advance to the ‘student_sender.py’ and send the cracked cleartext password. Note – if the hacker is successful, they should be able to log in to the receiver’s micro:bit without ever being told the password!

4. Practice adding salt to password:
 - a. Open ‘set_pw_5.py’ in the editor again. Edit the salt variable with a word, such as “enigma”. Do the same to the ‘authen_5.py’ file. The salt must be identical in both. Notice the programmer is adding the salt, not the user.

```

EDITOR: SET_PW_5
PROGRAM LINE 0019
if password1 == password2:
# add salt here
salt="enigma"
password_hash = sha_hash(password1)
if salt!="":
password_hash+=salt
password_hash=sha_hash(password_hash)
write_file("password.txt", password_hash)

EDITOR: AUTHEN_5
PROGRAM LINE 0014
for n in range(3):
password = input("Enter password: ")
# add salt here
salt="enigma"
test_hash = sha_hash(password)
if salt!="":
test_hash+=salt
test_hash=sha_hash(test_hash)
  
```

- Open 'set_pw_5.py' in the editor again and run to set the password using the same one of the ten most common passwords. This time, however, the salt will be added to the password.
- Open 'pract_3_5.py' in the editor again and run the program to display the hash stored on your micro:bit. Even though the password is the same as the password in 2.b, do you notice it has a different hash than the one in 2.c?
- Use the rainbow table 'rbt' to look up the password corresponding to the hash as you did in practice 1.e. To use the table, type `rbt[hacked_hash]` in the Python shell at the `>>>` prompt on the next page. Note – the search of 'rbt' should fail with an error message "**KeyError:**" because there is no hash index in the dictionary even though you used one of the ten common passwords.

Code it

Sender role

```

EDITOR: SEND_5
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_5
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_5
PROGRAM LINE 0001
from microbit import *
from mb_radio import *
from mb_file import *
from rbt_5 import *
stolen_hash=""
radio.on()
radio.config(length=250, channel
            =12,power=6,group=1)
disp_clr()
print("Man-in-the-middle attacki
ng...")

```

Go further

- Each team member should play each role and try to hack the other's password.
- Each team member should Repeat step 4 using a different salt but the same common password. Next, switch micro:bits within the group and test if your platform (calculator) will open their micro:bit. Note- everyone on the team has used the same password, yet authentication does not work on anyone else's calculator because each calculator uses a different salt.

Check your understanding

- A rainbow table is a hacker resource used to gain unauthorized access to an account or device.
- A rainbow table is a Python dictionary of hashes with the corresponding cleartext password.
- Adding salt to a password is a method a cybersecurity programmer can use to add an additional layer of protection from hackers for users who use the same password on different platform accounts.
- Frequently changing your password and incorporating many infrequently used characters helps to protect your password from appearing in a hacker's rainbow table.
- Avoid using the same password on multiple accounts.

Help

- Check that everyone on the team is using their assigned group number.
- Ensure the receiver and hacker run their programs and wait before the sender transmits the message.
- Ensure passwords are successfully set on each micro:bit.
- Remember, the sender is trying to log in to the receiver's micro:bit and must send the password for that micro:bit.
- A student must use the same salt when setting the password as when authenticating the password.

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
prac_1_5.py	Select one of the ten common passwords at random and store the hash.
prac_2_5.py	Recalls the practice hash and practices using the rainbow table dictionary.
prac_3_5.py	Recalls and displays password hash set from 'set_pw_5.py'.
set_pw_5.py	It prompts for a password and saves the hash in the 'password.txt' file on the micro:bit.
authen_5.py	Compares the hash of the entered password with the hash stored on the micro:bit.
send_5.py	Sends hashed password to receiver for authentication.
recv_5.py	Receives hashed password and authenticates.
hack_5.py	Man-in-the-middle attack to snatch the hash from the sender.
rbt_5.py	Python dictionary with ten common passwords indexed by their hashes.
HASHING.8XV	SHA-256 hashing module