

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, 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 hashes. **Rainbow tables** are 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.
- A rainbow table is an extensive Python **dictionary** of cleartext passwords indexed with the corresponding computed hashes.
- Multiple computers precompute rainbow tables and take a long time to compile. They are available for hackers to download on darknet websites.
- Searching a rainbow table is 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 helps to reduce your chances of it appearing in a rainbow table.
- A **salt** is a key phrase added to all passwords on a particular platform. The salt added to the password differs for each platform, such as a web banking or shopping account. Salts are added to the password when setting it and during authentication.
- If a person uses the same password across several platforms, the stored hash will differ on each platform because they are hashed with different salts.

What will you do?

- Practice searching a rainbow table:
  - Advance to the page with “hacking\_password\_simulation.py” and run the program
  - a random hash will be stolen from the 'practice\_password.txt' file stored on the micro:bit and saved in the Python shell in the variable named 'hacked\_hash.'
  - Use the rainbow table named 'rbt' to look up the password corresponding to the hash. To use the table, type `rbt[hacked_hash]` in the Python shell at the **REPL >>> prompt** on the next page. Hint – use the Nspire’s [var] key to select the variable “hacked\_hash” from a menu.
  - Repeat steps 1b and 1c to practice using the rainbow table a few more times.
- Set a password on your micro:bit.
  - Choose one of the ten common passwords** included in the rainbow table of this activity.

Password	<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>

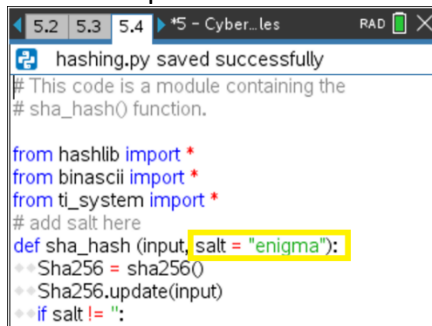
  - Advance to the page with “set\_password.py” and run the program to set the password of your choice on your micro:bit.
  - Advance to “read\_password\_hash.py” and run the program to display the hash stored on your micro:bit.
  - Repeat step 1c to hack the password you just set in 2a. Note - the RBT search should return the password you set in 2b. Do you understand how RBT is used to find the password of a hash?
  - Advance to “authentication.py” and run the program to test your new password.

### 3. Remote login:

- The **receiver**
  - **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**
  - 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 microbit remotely.
- The **hacker**
  - 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.
  - Use the stolen hash and the rainbow table, as you did in step 1c, to look up the receiver's password from the intercepted hash.
- Once the hacker has cracked the receiver's password, the receiver should run the 'student\_receiver.py' program again to test whether the hacker can break into your micro:bit.
- 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. Advance to the page with the module "hashing.py" and add an additional parameter, the salt, to the "Sha256" function. Add a salt that is a simple word, such as "enigma." Note: This should only be done on the student\_sender.py and the student\_receiver.py programs. Remember, this is a technique to block hackers.



```
hashing.py saved successfully
# This code is a module containing the
# sha_hash() function.

from hashlib import *
from binascii import *
from ti_system import *
# add salt here
def sha_hash(input, salt = "enigma"):
    Sha256 = sha256()
    Sha256.update(input)
    if salt != "":
```

- b. After editing the function, you must recompile the code by pressing the blue [ctrl] key and then the [B] key. Once completed, the message "hash.py saved successfully." If the file did not save successfully, check the syntax for errors and ensure the comma separates the parameters and the salt has quoted.
- c. Advance back to the page with the program "set\_password.py" 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.
- d. Advance to "read\_password\_hash.py" and run the program to display the hash stored on your micro:bit. Even though the password is the same as the password in 2b, do you notice it has a different hash than the one in 2c?

- e. Use the rainbow table named 'rbt' to look up the password corresponding to the hash. 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. Remember, the salt added to the sender and receiver changed the hash and thus is no longer in the rainbow table.

## Code it

### Sender role

```

5.1 5.2 5.3 ▶ *5 - Cyber...les RAD
student_sender.py saved successfully
from microbit_radio import *
from hashing import *
# The sender must use the password of the
# receiver's micro:bit.
channel = 1
group = 1
clear_history()
password = input("Enter password: ")
password_hash = sha_hash(password)
tx(password_hash,channel,group)

```

### Receiver role

```

5.1 5.2 5.3 ▶ *5 - Cyber...les RAD
student_receiver.py saved successfully
from microbit_radio import *
from hashing import *
# Secretly share your password with the sender.
# Keep it private from the hacker.
channel = 1
group = 1
clear_history()
test_hash = rx(channel,group)
authentic_hash = read_file("password.txt")
if test_hash == authentic_hash:
    display.show(Image.HAPPY)

```

### Hacker role

```

5.1 5.2 5.3 ▶ *5 - Cyber...les RAD
student_hacker.py saved successfully
from microbit_radio import *
from rainbow_table import *
# The hacker will use the rainbow table to find
# the password that was sent to the receiver.
channel = 1
group = 1
clear_history()
hacked_hash = rx(channel,group)
print("hash received from radio: ",hacked_hash)
# On the next page type rbt[hacked_hash]
# at the Python shell prompt >>>.

```

## 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 programmer can use to add an additional layer of protection from hackers.
- Frequently changing your password and incorporating many infrequently used characters helps to protect your password from appearing in a hacker's rainbow table.

## 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.