

### How it works

- A substitution **cipher** uses the same **alphanumeric** and punctuation **characters** in both **plaintext** and **ciphertext**.
- To **encipher** a message, each plaintext character's alphabetical position is shifted (translated) to the right or left. The character at that new position becomes the ciphertext character.
- To **decipher** a ciphertext, the translation process is reversed.
- The same key must be used to encipher and decipher a message.
- The table below shows the characters' positions in the alphabet with two punctuation marks.

| character | 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  | space | !  |
|-----------|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|----|
| position  | 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 | 26 | 27    | 28 |

- Example: encipher the character "H" in the plaintext "HELLO" using a key = 3 and a right shift:
  - Locate 'H'; it is the 8<sup>th</sup> character in the table.
  - Add the key of 3 to the position;  $8 + 3 = 11$
  - Locate position 11; it is the character 'K.'
  - The first letter of the ciphertext is 'K.'
  - If you come to the end of the table, continue counting from the beginning.
  - Repeat the previous steps for the remaining characters; the result is the ciphertext 'KHOOR'.
- A substitution cipher can be cracked performing a **character frequency analysis** of the the ciphertext. In common plaintext writing the characters of the alphabet each have a frequency on average in large text. In the English language a space (**ASCII 32**) is most common character and 'E' is most the most common letter. Since 'E' is the 5<sup>th</sup> letter in the alphabet, by counting the frequency of each character in the ciphertext, the key can be deduced. For example if the most frequent character in the ciphertext is 'K', then there is a good chance, 'K' (position 11) is being substituted for 'E' (position 5) and the shift key is 6 because  $11 - 5 = 6$ . This type of frequency analysis requires a large ciphertext to assume the most frequent ciphercharacter code for 'E'.

### What will you do?

- Practice Caesar ciphering:
  - using a right-shift and key = 6 along with the table above. Fill in the blank ciphertext space below.  
plaintext: HAIL CAESAR!                      ciphertext: \_ \_ \_ \_ \_
  - using a right-shift and key = 6 along with the table above. Fill in the blank plaintext space below.  
plaintext: \_ \_ \_ \_ \_                      ciphertext: ICHKXEOYEL TF
  - Advance to the page with "check\_practice\_1a.py" and run the program to check your encipherment of 1.a. Read the comments in the Python code to help you understand what the code is doing. Do the coding steps match your steps to encipher the plaintext in practice 1a?
  - Advance to the page with "check\_practice\_1b.py" and run the program to check your decipherment of 1b. Read the comments in the Python code to help you understand what the code is doing. Do the coding steps match your steps to decipher the ciphertext in practice 1b?
  - What happens to the ciphertext if you change the key to a different number and re-run the program?
- Practice using the caesar\_cipher module:
  - Advance to the page with "student\_encipher.py." Run the program to encipher the plaintext message. The ciphertext is automatically saved to the OS. Note - This program is generalized to

## TI-Nspire CX II

encipher a message using the 'ceaser\_cipher.py' module at the end of the file. Modules help chunk commonly used code into a form that can be added to new programs using the included statement.

- b. Advance to the page with "student\_decipher.py." Run the program to decipher the ciphertext saved in the OS from the previous program. Try changing the plaintext message and rerunning the two programs.

### 3. Practice using the character frequency analysis:

- a. Advance to the page with 'chr\_frequency\_practice.py' and run the program. Press the [var] key, select 'code\_to\_chr()' from the menu, and enter an ASCII code for the argument. For example, >>> code\_to\_chr (72) returns the letter "H". Try a few more ASCII codes to see which characters they represent.
- b. When you ran the program, the frequency of each character in the ciphertext of the first paragraph of Charles Dickens's novel A Tale of Two Cities was counted. Two lists contain the ASCII character code of each letter in the text and the other, the frequency of that character within the text. \* Note that the key is set to zero.
- c. Advance to the data and statistics graph page. The graph has the character code (chr\_code) on the horizontal axis and the frequency (f) on the vertical axis. Select [menu] -> 5: Window/Zoom -> 2: Zoom - Data will fit the data into the window.
- d. Hover the cursor over the most frequent character. The ASCII character code and frequency are displayed (ASCII code, frequency) above the cursor. Next, hover over the second most frequent character. Remember these two character codes for the next step.
- e. Navigate back one page to the Python shell created when running the character count practice program. At the Python REPL >>> prompt on that page, press the [var] key, select 'code\_to\_chr()' from the menu, and enter an ASCII character code for the most frequent character as the argument of the function. Repeat for the character code of the second most frequent character.
- f. What letter is the most frequent in the text?
- g. Navigate back to the page with 'chr\_frequency\_practice and change the key to three, i.e., key=3. Repeat steps "3.a" through "3.d". Notice the ASCII character codes for the most frequent and second most frequent characters have changed; they have shifted three places to the right. Remember, the ASCII character code has not shifted three places; instead, the character in the chr\_set list defined within the program has shifted three positions in the list.

### 4. Texting an encrypted message:

- The **receiver**
  - Select the 'student\_receiver.py,' change the group to the assigned number, and then run the program **before** the sender has run theirs.
- The **sender**
  - select the 'student\_sender.py,' edit the message string, change the group to the assigned number, and then run your program **after** the receiver and hacker have started theirs.
- The **hacker**
  - a. select the 'student\_hacker.py,' change the group to the assigned number, and then run the program **before** the sender has run theirs.
  - b. After the man-in-the-middle attack steals the ciphertext, repeat the process from the practice in step 3 above.

## TI-Nspire CX II

1. Press the [var] key and select count\_chrs\_hack(), press the [var] key again, and select 'ciphertext.' It should look like this: >>>count\_chrs\_hack(ciphertext), then press enter to count the characters in the stolen message.
2. Advance to the data and statistics graph page. Repeat the analysis as outlined in 3.c through 3.e. through 3.g. Note: the cipher key is the number of letters the plaintext is shifted in the ciphertext.
3. Test your analysis of the key by running the student\_receiver program. At the prompt, enter the hacked key.
4. Have the student\_sender send the message again. Be sure they do not announce the key to the group. Did you hack the message?

## Code it

### Sender role

```

3.1 3.2 3.3 ▶ *3 - Cyber...ar! RAD [icon] X
student_sender.py 1/11
from microbit_radio import *
from caesar_cipher import *
# secret key must be same as receiver
message = "The gold is in the cookie jar!"
key = 42
channel = 1
group = 1
enciphered_text = encipher(message,key)
clear_history()
print("\nenciphered_text = ",enciphered_text)
tx(enciphered_text,channel,group)

```

### Receiver role

```

3.1 3.2 3.3 ▶ *3 - Cyber...ar! RAD [icon] X
student_receiver.py 1/11
from microbit_radio import *
from caesar_cipher import *
# secret key must be same as sender
key = 42
channel = 1
group = 1
clear_history()
enciphered_text = rx(channel,group)
print("\nreceived: ", enciphered_text)
deciphered_text = decipher(enciphered_text,key)
print("\ndeciphered_text = ",deciphered_text)

```

### Hacker role

```

3.1 3.2 3.3 ▶ *3 - Cyber...ar! RAD [icon] X
student_hacker.py 1/11
from microbit_radio import *
from caesar_cipher import *
# secret key must be unknown to the hacker
key = 1
channel = 1
group = 1
clear_history()
enciphered_text = rx(channel,group)
print("\nreceived: ", enciphered_text)
message = decipher(enciphered_text,key)
print("\ndeciphered: ",message)

```

## Go further

- Try a different role in your team.
- Change the key. Is the ciphertext the same?

## Check your understanding

- Ciphers are used to **obfuscate** (hide) plaintext messages from hackers.
- A key is required in the algorithm to translate the plaintext characters to the ciphertext characters.
- The sender and receiver must use the same key.
- Frequency analysis is a technique that can be used to decipher messages.

## 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 the sender and receiver use the same key and it is private from the hacker.