CS4003701 資訊安全導論 Introduction to Information Security
HW1 Report

Team members : B10832008 蔡芸軒(encrypt)、B10832018 官澔(decrypt)

[Structure tree]

```
110-Information-Security
| ----HW1
      | ---- encrypt.py
      | ---- decrypt.py
      | ---- Report.pdf
| ----HW2
| ----HW3
```

[encrypt]

As required, we use `sys.argv` to read arguments. And we get the `-m` argument to see what method we should implement, then call the function with the same name, passing parameters `-i` (represent plaintext) and `-k` (represent key). We have five encryption methods : `caesar`, `playfair`, `vernam`, `rail_fence`, and `row_transposition`.

`caesar` :

Iterate through the string, use `ord()` to get ascii and the offset from lowercase 'a' of each character, add key to it then mod 26, finally use `chr()` to get the corresponding uppercase character.

`playfair` :

Contribute the `key_matrix` by first filling the character in key string in turn then a to z that haven't appeared yet. We also contribute an 2D array `key_row_column` to cache the row and column of each towel. After that, iterate two characters each time through the string, inserting `'X'` if two characters are the same. Else, seeing whether they share the same row or the same column or not, just follow the rule, do according process.

`vernam` :

Contribute the `key` from the original `key` and `plaintext`, making the `key` string the same length as the `plaintext`. Iterate through two strings at the same time, use `ord()` to get the offset from lowercase `'a'` of each corresponding two characters, add up them then get a new character.

`railfence` :
Set an array with length “key”. Iterate through the plaintext and push the characters into i^{th} array, with i be 0 to “key”, modifying in zig-zag.

`row` :
Use the new key to cache the appearance sequence of the column, and contribute the matrix. Then just iterate through the key, getting the corresponding column, and iterate through each row to get the character.

[decrypt]
The program implemented 5 decrypt methods: caesar, railfair, vernam, rail_fance and row_transportation. We use sys.argv, the same as encrypt, to handle arguments.
-m {method} : choose the method to decrypt the ciphertext.
-i {ciphertext} : put the input encrypted by the method above
-k {key} : place the key here

caesar:
Use ord() to get the ASCII code and place a~z value to 0~25. Reverse each character by key and mod by 26. to get plaintext. Finally return the lower case of the plaintext.

playfair:
First Reconstruct the key matrix using keywords. Then use the same method as encryption but on the opposite way. Then We get the plaintext. Return the lower case of the plaintext.

vernam:
Get the ASCII code of both ciphertext and keyword. Then do the XOR bitwise operation to get the plaintext. Return the lower case of plaintext.

rail fence:
First I count how many cycles does the ciphertext have, then figure out how many words will be placed in each row. Then we can iterate ciphertext and put the text back into the plaintext string.

row:
Use the Keyword to put the ciphertext into the text board. then iterate the textboard to get the plaintext. Return the lower case of plaintext.