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 ith 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]