Date handed out: See ODTUClass
Date submission due: See ODTUClass

Classical Cipher (Vigenere+Columnar)

Purpose

The main purpose of this programming assignment is to revise and gain hands-on experience on the classical ciphers. A classical cipher is a type of cipher that was used historically but for the most part, has fallen into disuse. In contrast to modern cryptographic algorithms, most classical ciphers can be practically computed and solved by hand. However, they are also usually very simple to break with modern technology.

Introduction

Cryptography is the science and study of disguising messages so that only certain people can see through the disguise. Before beginning, let's define some terms. An original message is known as the **plaintext**, while the coded message is called the **ciphertext**. As seen in **Fig. 1**, the process of converting from plaintext to ciphertext is known as **encryption**; restoring the plaintext from the ciphertext is **decryption**.

In the encryption process, the algorithm, known as **cipher**, takes the plaintext and a **key** (aka password) as input, applies the encryption algorithm and outputs the ciphertext. Similarly, the decryption process takes a ciphertext and the key, applies the decryption algorithm, and outputs the plaintext.

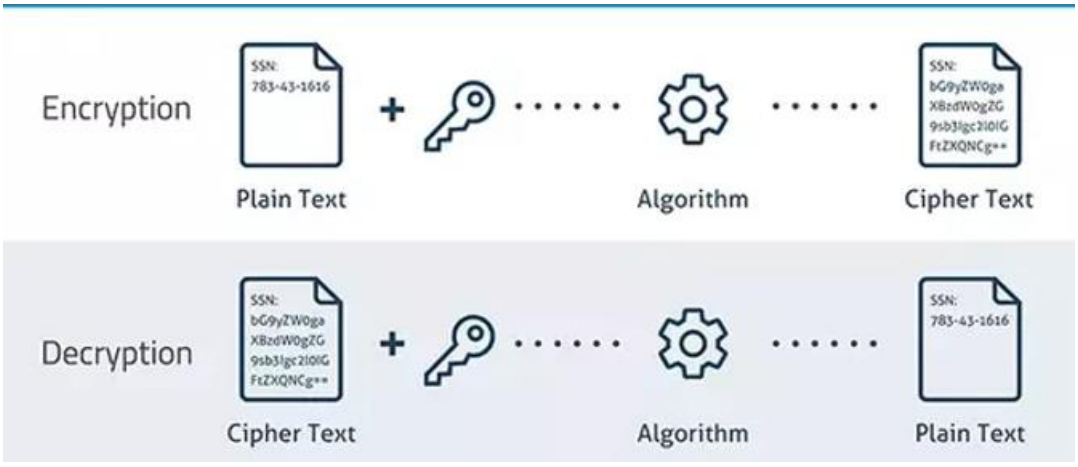


Fig. 1. Encryption and Decryption Process

The encryption and decryption algorithm are explained below in detail.

Assignment – General Look

The programming assignment combining Vigenère cipher and a transposition cipher is as follows:

- Write a function that can encrypt and decrypt messages using the Vigenère cipher. The Vigenère cipher is a polyalphabetic substitution cipher that uses a keyword to choose which

alphabet to encipher letters. The program should take as input a plaintext message and a keyword, and output the ciphertext message. The program should also be able to reverse the process, given a ciphertext message and a keyword, and output the plaintext message.

- Write a function that can encrypt and decrypt messages using the columnar transposition cipher. The columnar transposition cipher is a simple transposition cipher in which letters are arranged in rows and the columns are transposed according to a key. The program should take as input a plaintext message and a key (the same with the previous cipher), and output the ciphertext message. The program should also be able to reverse the process, given a ciphertext message and a key, and output the plaintext message.

- Write a program that can combine the Vigenère cipher and the columnar transposition cipher to create a more secure encryption scheme. The program should first apply the Vigenère cipher to the plaintext message using a keyword, then apply the columnar transposition cipher to the resulting ciphertext using another key. The program should also be able to reverse the process, given a ciphertext message and both keys, and output the plaintext message.

The programs can be written in any programming language of your choice. You should test your programs with different messages and keys, and compare the results with online tools or other sources.

Encryption Algorithm

Encryption will be carried out in two phases.

First Phase:
First begin with a polyalphabetic substitution. Assume a sequence of plaintext letters $P = p_0, p_1, p_2, \ldots, p_{n-1}$ and a key consisting of the sequence of letters $K = k_0, k_1, k_2, \ldots, k_{m-1}$, where typically $m < n$. The sequence of ciphertext letters $C = c_0, c_1, c_2, \ldots, c_{n-1}$ is calculated as follows:

You will use a conversion table (see **Fig. 2** for an illustration) to encrypt the given plaintext. The table will be given to you in form of a two-dimensional array (see the attachment). The rules are very simple. In encrypting plaintext, the cipher letter (c_0) is found at the intersection of the column headed by the plaintext (i.e. p_0) and the row indexed by the key letter (i.e. k_0). And then you will continue with the next letter in the plaintext and in the key until all the letters in the plaintext is converted to the ciphertext letter. As the length of the key is shorter than the length of the plaintext, the key will be repeated.

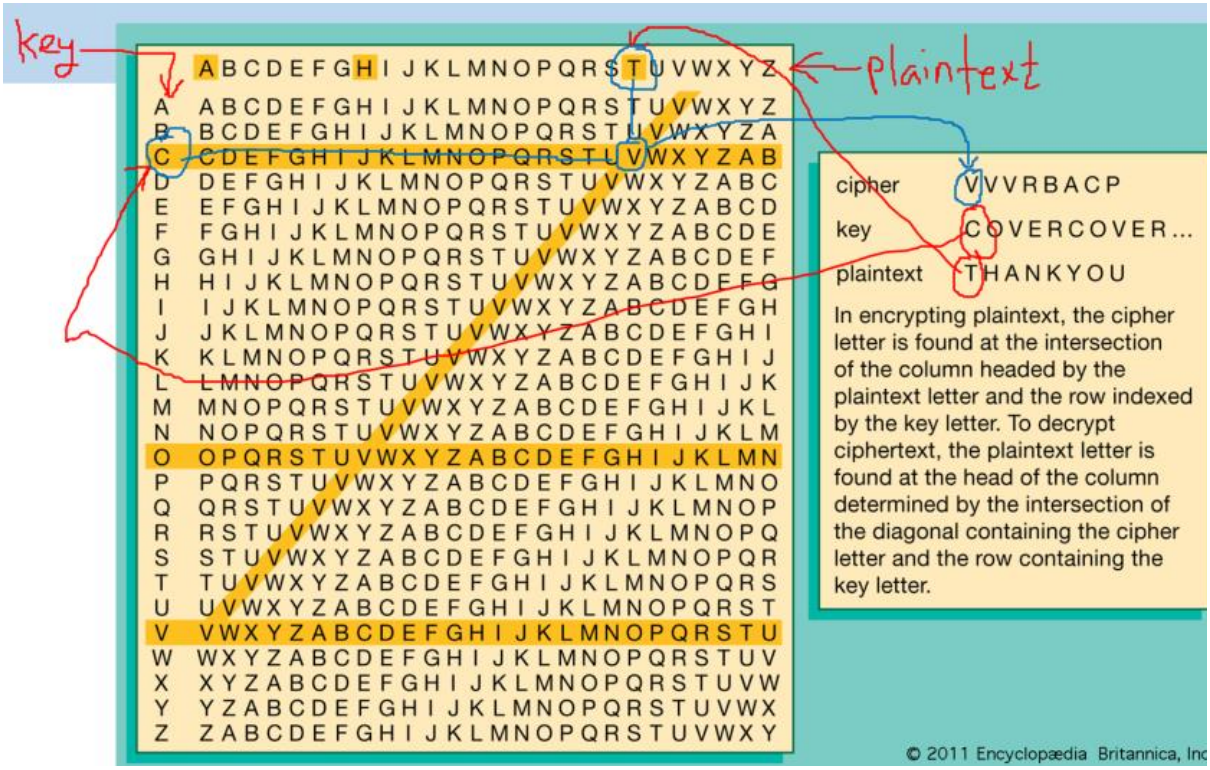


Fig. 2. Encryption and Decryption Conversion Table (taken from Britannica) – This is just for illustration of the algorithm. The conversion table you will use in this assignment is given in the attachment.

For example:
First get the text from the user.
Input: *C IS FUN*
Trim the spaces from the input and obtain the plaintext.
Plaintext: *CISFUN*
Get the key from the user
Key: *PEACE*

First take the first character of plaintext (i.e. C) and take the first character of the key (i.e. *P*) and find the intersected character (i.e. *R*) in the table. And repeat this step until you convert all the letters in the plaintext. Notice that the key is repeated if necessary.

Plaintext	C	I	S	F	U	N
Key	P	E	A	C	E	P
Generated Ciphertext	R	M	S	H	Y	C

After the conversion process, you obtain the output (i.e. ciphertext for the first phase).
Ciphertext: *RMSHYC*

Second Phase:
In this phase. You will use the output of the first phase (e.g. *RMSHYC* for the example) as input. And then apply the columnar transposition cipher as follows:

1. Use the keyword as the key and determine its length (let's call it k).

2. Write the input text in rows of length k, adding a padding character (e.g., zero) if needed to fill the last row.
3. Reorder the columns based on the alphabetical order of the letters in the keyword, and assign each column a number according to its position in the permuted order.
4. Read off the ciphertext by columns in the permuted order.

Example

The key for the columnar transposition cipher is a keyword e.g. `GERMAN`. The row length that is used is the same as the length of the keyword. To encrypt a piece of text, e.g.

```
defend the east wall of the castle
```

we write it out in a special way in a number of rows (the keyword here is `GERMAN`):

```
G E R M A N
d e f e n d
t h e e a s
t w a l l o
f t h e c a
s t l e x x
```

In the above example, the plaintext has been padded so that it neatly fits in a rectangle. This is known as a regular columnar transposition. An irregular columnar transposition leaves these characters blank, though this makes decryption slightly more difficult. The columns are now reordered such that the letters in the key word are ordered alphabetically.

```
A E G M N R
n e d e d f
a h t e s e
l w t l o a
c t f e a h
x t s e x l
```

The ciphertext is read off along the columns:

```
nalcxehwttdttfseeleedsoaxfeahl
```

* Example is taken from http://practicalcryptography.com/ciphers/classical-era/columnar-transposition/

Decryption Algorithm

Decryption is the reverse operation of encryption. Similarly, there will be two phases corresponding to each encryption phase in reverse order (see Fig. 3).

First Phase:

1. Ask the same keyword as the key and determine its length (k).
2. Divide the length of the ciphertext by k to get the number of rows (let's call it r).
3. Write the ciphertext in columns of length r, leaving some cells blank if needed to fill the last column.
4. Reorder the columns based on the original order of the letters in the keyword, and assign each column a number according to its position in the original order.
5. Read off the plaintext by rows in the original order.

Second Phase:

This is the reverse operation of encryption phase 1. To decrypt the text, the plaintext letter is found at the head of the column determined by the intersection of the diagonal containing the cipher letter and the row containing the key letter using the conversion table provided (see Fig. 2).

Assumptions

1. Consider only English letters (i.e. 26 letters) in plaintext, in ciphertext, and in key, no need to consider other characters (e.g. digits, punctuation characters, etc.).
2. Erase any space found in the input.
3. Convert the text uppercase first.

Algorithm Explained

The program flow is depicted in **Fig. 3**.

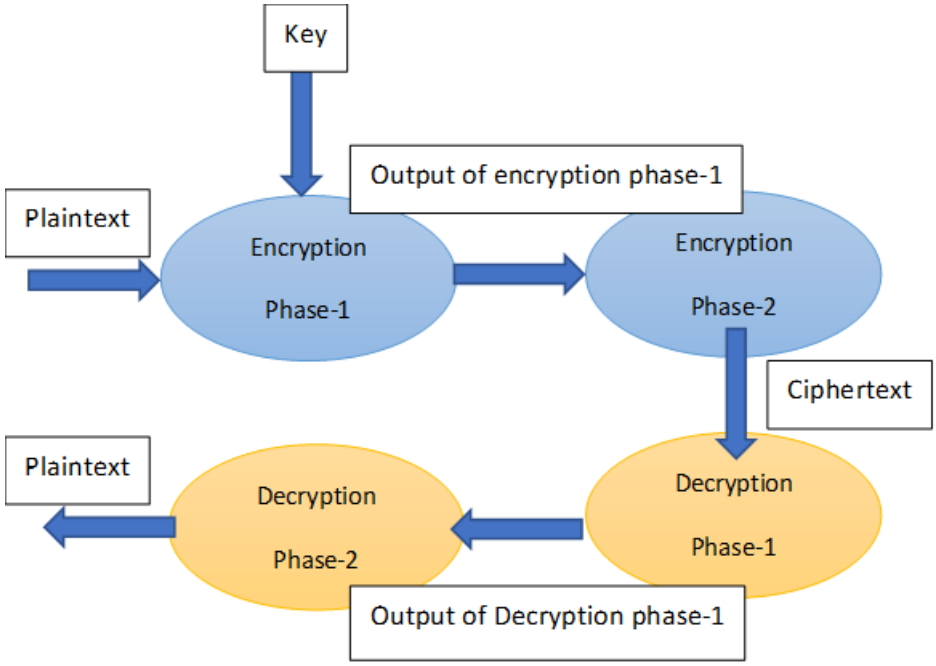


Fig. 3. Program Flow

Prepare the input string (i.e. plaintext for encryption or ciphertext for decryption):

1. Read the input from the user.
2. Trim whitespace (e.g., space).
3. Convert all the letters to uppercase.

Prepare the key:

1. Read the key from the user (both for encryption and decryption).
2. If the key length is less than the length of the input string, then repeat the keyword to match the length of the input string.

Encryption:

1. Apply the first phase for the encryption. **Hint:** The columns and rows are shown with letters in the conversion table. Therefore, you need to find the index (the order of the letter in English alphabet) of the letter. In order to match the letter and its corresponding order in the alphabet, you may use ASCII table as a trick. As all our letters in the uppercase form, you may obtain the order of the letter in English alphabet by subtracting 65, whereas the ASCII code

of 'A' is 65. For example, if you subtract 'A' - 65, you obtain 0, which is the index for the character 'A' to determine the column or row index in the conversion table.
2. Apply the second phase for the encryption. Use the output of phase 1 as the input for the phase 2. If required, add the padding characters.
3. Print the ciphertext.

Decryption:

1. Apply the first phase of the decryption algorithm (, which is the reverse operation of phase-2 of encryption).
2. Apply the second phase of the decryption algorithm (, which is the reverse operation of phase-1 of encryption).
3. Print the plaintext.

Programming Requirements

- You may use any programming language to implement. The interface will be similar as in the given **Sample Run** subsection below.

- The conversion table for the encryption phase-1 and the decryption phase-2 is provided as a 2-D array. See the attachment.

- The following functions must be implemented:

    - **Menu**: This will print the menu as seen in the sample run below. It will take the user selection. The menu will repeat until the user selects exit option.

    - **Encrypt**: This will encrypt the plaintext using the key provided by the user and it will return the ciphertext using the algorithm defined above.

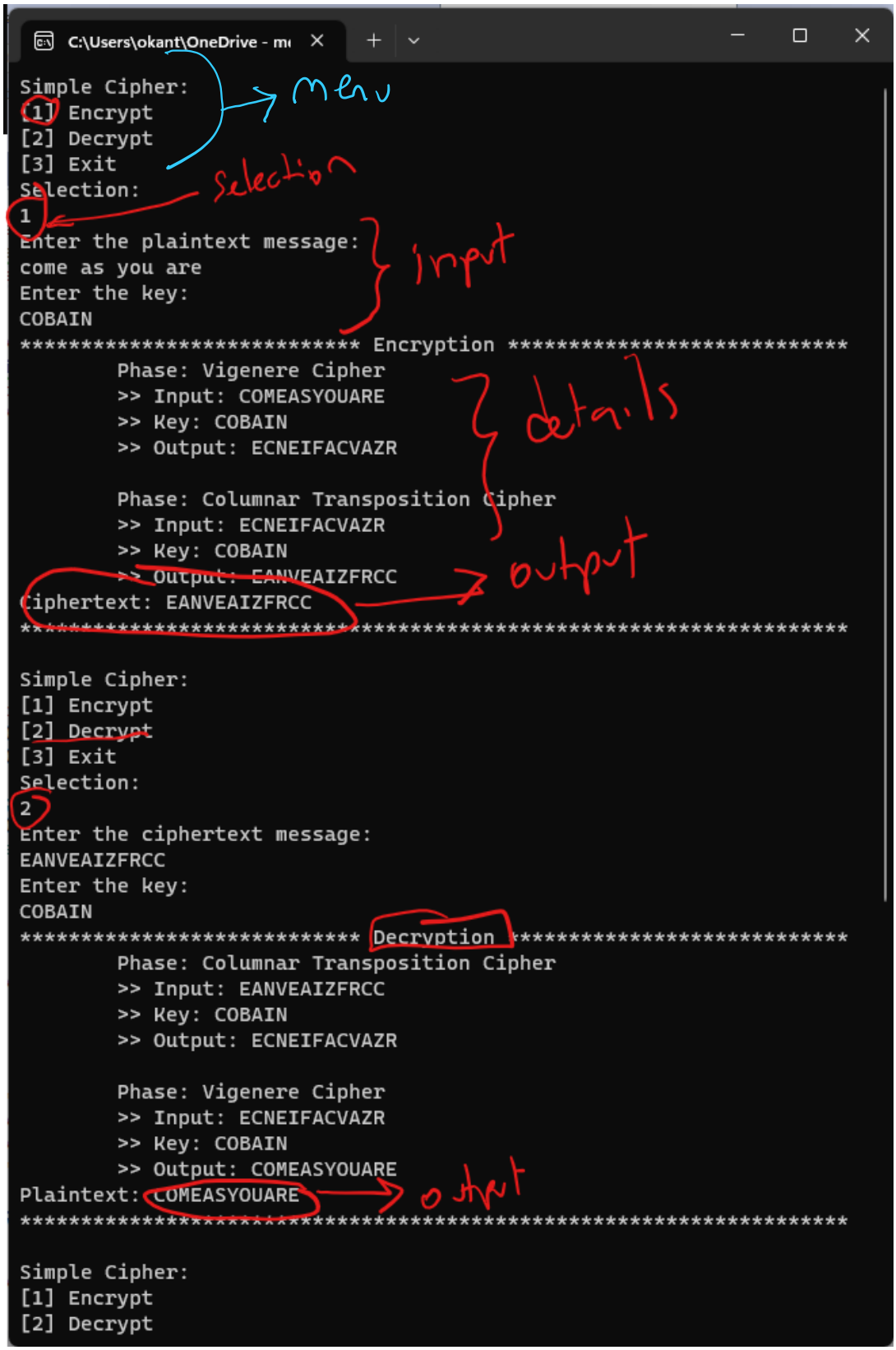
    - **Decrypt**: This will decrypt the ciphertext using the same key in the encryption and it will return the plaintext using the algorithm defined above.

- And you may implement any other helper functions such as trimming the spaces in a string, getting the length of a string etc. Some useful functions you may consider implementing:

    - A function that returns the length of a given string. Just count the letters in a loop until the null character at the end of the string.
    - A function that repeats the key. You will repeat key as long as the length of the input text. You may keep two indices in a loop as one goes through the length of the input text and the other resets in each repetition of the key.
    - A function to trim the space from the given string.

Important note: Go through the sample run carefully. It's very helpful to understand about this practical assignment. Take a look at the grading distribution to get idea which parts of your codes will be evaluated strictly.

Sample Run


C:\Users\okant\OneDrive - mı
Simple Cipher:
[1] Encrypt
[2] Decrypt
[3] Exit
Selection:
1
Enter the plaintext message:
come as you are
Enter the key:
COBAIN
********** Encryption **********
Phase: Vigenere Cipher
>> Input: COMEASYOUARE
>> Key: COBAIN
>> Output: ECNEIFACVAZR
Phase: Columnar Transposition Cipher
>> Input: ECNEIFACVAZR
>> Key: COBAIN
>> Output: EANVEAIZFRCC
Ciphertext: EANVEAIZFRCC
****
Simple Cipher:
[1] Encrypt
[2] Decrypt
[3] Exit
Selection:
2
Enter the ciphertext message:
EANVEAIZFRCC
Enter the key:
COBAIN
********** Decryption **********
Phase: Columnar Transposition Cipher
>> Input: EANVEAIZFRCC
>> Key: COBAIN
>> Output: ECNEIFACVAZR
Phase: Vigenere Cipher
>> Input: ECNEIFACVAZR
>> Key: COBAIN
>> Output: COMEASYOUARE
Plaintext: COMEASYOUARE
****
Simple Cipher:
[1] Encrypt
[2] Decrypt
menu
selection
input
details
output
output

Grading

Your program will be graded as follows:

Grading Distribution	Mark (100)	Remarks
Menu function	5	
Encryption operation Phase-1	20	
Encryption operation Phase-2	20	
Decryption operation Phase-1	20	
Decryption operation Phase-2	20	
Other useful functions (trim, padding, uppercase, etc.)	5	
Readme.txt and demo video	10	

Rules

Please make sure that you follow the restrictions for the assignment as follows:

- Follow the mentioned programming requirements.
- If your program does not compile, then you will get 0.
- Strictly obey the input/output format. Do not print extra things.

Submission

You need to submit a ZIP file (firstname_lastname.zip, e.g., okan_topcu.zip) including the following:

1. Your source codes.
2. Your executable.
3. *Readme.txt* file for your source code. This should include a short description of your program and **it should explain how to compile and run your code**, please include your name, surname, and student id at the top.
4. A short video showing the execution demo. You may provide a link to an outside site (e.g., YouTube)

To submit your assignment, simply select the appropriate assignment link from the ODTUCLASS page. Upload your zip file and click submit (clicking send is not enough). Please make sure all source files are included in your zip file when submitted. A program that does not compile as submitted will be given 0 points. Only your final submission will be graded. Remember there is no late submission for this assignment.

Academic Honesty

You need to submit your own solution. Your source code will be investigated for neatness and cheating. All the programming assignments will be required to be implemented individually and any code sharing will be considered as cheating. Please note that if you are caught cheating, you can get zero from this assignment and also from all the others. See the course syllabus for further details and clarification.

Attachments

Table.txt: Two-Dimensional Array Definition (just in case)