# *EECE 455: Cryptography and Network Security*

# *Project Report: Two Square Playfair Cipher*

## *Fall 2023-2024*

*Done by: Reem Arnaout (202202256) & Aya Al Baba (202200368)*

*Professor: Khalil Hariss*

# List of Contents:

## Overview:

The Two-Square Playfair Cipher is a traditional encryption technique that was created as an improvement of its parent technique (The Playfair Cipher). In this report, we will go over some of the history and qualities behind this cipher, how it works, how we implemented it in software, and the difficulties we faced in this process. We also provide a shallow explanation of how machine learning can be intertwined with the Two Square Playfair Cipher to almost eliminate its deficiencies. In the end of the report, there is an appendix that includes the detailed code that we created for this project.

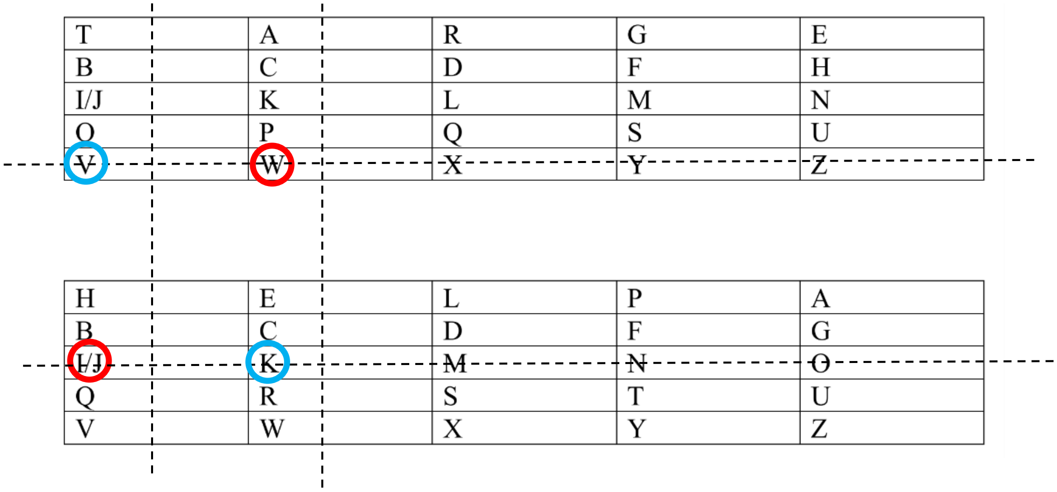
## o Background and Explanation of the Two Square Playfair Technique:

Invented by Sir Charles Wheatstone in 1854, the Playfair Cipher is the first cipher to encrypt pairs of letters simultaneously. In 1901, the Two Square Playfair Cipher was introduced by Félix Delastelle in his book “Traité Élémentaire de Cryptographie”. This symmetric encryption technique is considered superior to its parent technique due to its lower susceptibility to frequency attacks.

The cipher uses two 5x5 matrices aligned below each other. After attaining two keywords, enter the letters into the two matrices as follows: (0,0), (0,1), (0,2), etc. (Without repeating the entered letters). Then, in each matrix, fill the remaining cells with the remaining letters of the alphabet. Read the plain text from left to right. Locate the first letter in the first matrix and the second letter in the second matrix and so on. If the letters fall under the same column, copy them as is into the encrypted text. Otherwise, form a rectangle with the diagonal obtained from the selected letters; the other two vertices are the respective encryption of the plain letters. Note that the Two Square Playfair Cipher eliminates the need to insert an “X” in between attached repeated letters in the plain text. A non-letter character should be copied as is into the encrypted text. We will examine later in detail how to deal with these exceptions. Below, a simple example of the technique is presented.

Simple Example:

Keyword 1: TARGET, Keyword 2: HELP, Plain text: we will rock



| | | | | |
|---|---|---|---|---|
| T | A | R | G | E |
| B | C | D | F | H |
| I/J | K | L | M | N |
| O | P | Q | S | U |
| V | W | X | Y | Z |

| | | | | |
|---|---|---|---|---|
| H | E | L | P | A |
| B | C | D | F | G |
| I/J | K | M | N | O |
| Q | R | S | T | U |
| V | W | X | Y | Z |

| Plain-text | we | wi | ll | ro | ck |
|---|---|---|---|---|---|
| Cipher-text | we | vk | ll | em | ck |

- Software Implementation:

*__Remark__: For the entirety of the software implementation, we make sure that all the letters processed in the algorithm and exited are capitalized. Plus, for simplicity, all J's present in the keywords and plain text are converted into I's. We also assume that the indices of the matrix and arrays start from 0.*

- Generating the Matrices:

  We start off the code with a "generate_matrix" function which is self-explanatory. It creates a matrix where "i" and "j" represent the rows and columns respectively. We use a dictionary to track the entered letters of the keywords. What's left out afterwards is then filled into the remaining cells of the matrix.

- General Idea of the Algorithm:

  The pattern that we realized related the plain text and cipher letters by their indices (i,j) as the following:

  - If $j_{plain(0)} = j_{plain(1)}$: append both letters to the encrypted text.
  - If $j_{plain(0)} \neq j_{plain(1)}$: the encryption of the first letter has the indices $(i_{plain(0)}, j_{plain(1)})$ while that of the second letter has the indices $(i_{plain(1)}, j_{plain(0)})$.
    In the simple example (page 4), W= (4,1), I= (2,0). Their encryption V= (4,0), K= (2,1) represent this relationship.

- Handling Exceptions:

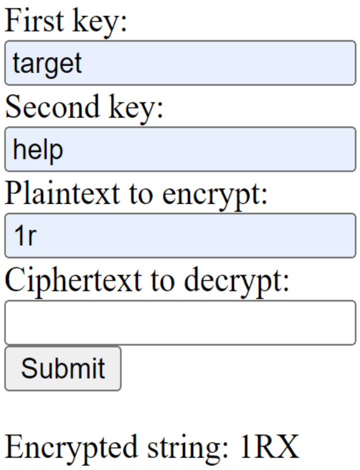
  - Non-letter Characters:

    Off the bat, keyword letters are not permitted to hold non-letters characters including spaces. If found, the code breaks showing an error statement.

    Plain text, however, can have non-letter characters. To deal with this exception, we create the function "search_target" which takes in a matrix and a target element. It then goes through all the elements of the matrix selected. If the target is found, it returns the indices of it; otherwise, it returns "None".

These characters are appended as is into the cipher text and aren't encrypted. This complicates the technique since it processes pairs of plain entries at once. The following rules apply:

1. If current entries are (None, Letter), append the None entry to the cipher text and move the plain list index by 1.
   <u>Example:</u>



First key: target

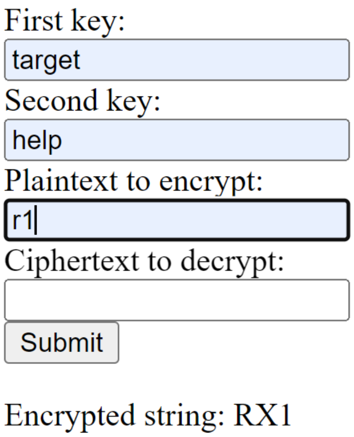
Second key: help

Plaintext to encrypt: 1r

Ciphertext to decrypt:

Encrypted string: 1RX

2. If current entries are (Letter, None), insert the letter "X" in between the entries and proceed with the algorithm.
   <u>Example:</u>



First key: target

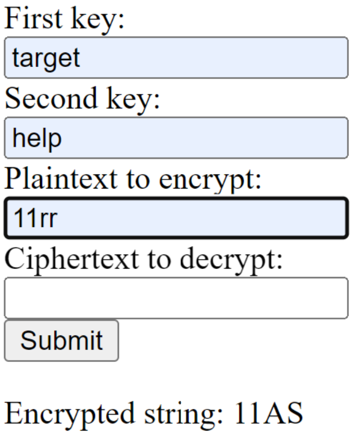
Second key: help

Plaintext to encrypt: r1

Ciphertext to decrypt:

Encrypted string: RX1

3. If current entries are (None, None), append both entries into the cipher text and move the plain list index by 2.
   Example:



First key:
target
Second key:
help
Plaintext to encrypt:
11rr
Ciphertext to decrypt:

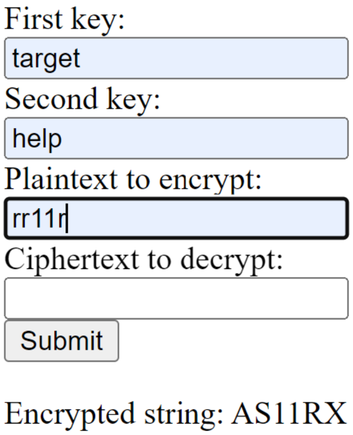
Submit

Encrypted string: 11AS

- Length of the Plain Text:

  Since the entries of the plain text are read as pairs simultaneously, this creates a bothersome error "string index out of range". In general, the length of the plain text including the inserted "X" s minus the number of non-letter characters should be even. In the code, k is a counter for the non-letter characters. The following rules apply when we arrive at the last plain entry:

1. If the last entry is a letter and the number of letters (after adding all the necessary "X" s) is odd, then append the plain text with an "X" to pair with this last entry for encryption.
   Example:

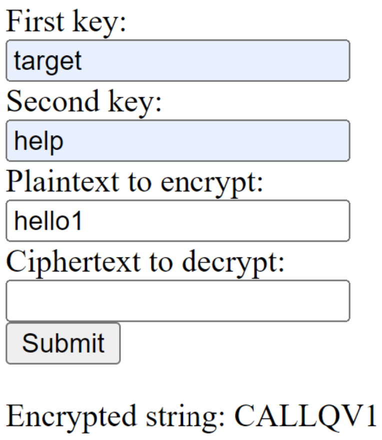


First key:
target
Second key:
help
Plaintext to encrypt:
rr11r
Ciphertext to decrypt:

Submit

Encrypted string: AS11RX

2. If the last entry is a not a letter, append this entry to the cipher text and exit the encrypting loop to avoid receiving an "out of range" error.
   Example:



First key: target

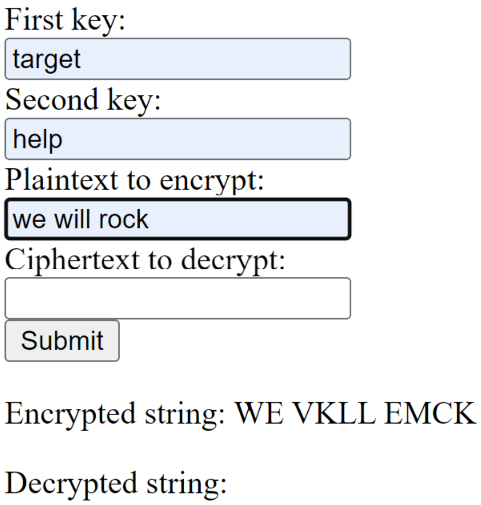
Second key: help

Plaintext to encrypt: hello1

Ciphertext to decrypt:

Encrypted string: CALLQV1

- Decryption:
  What makes the Two Square Playfair ciphers relatively simple is the decryption method being identical to the encryption method. Decryption follows the same steps and rules as encryption.
  Example:



First key: target

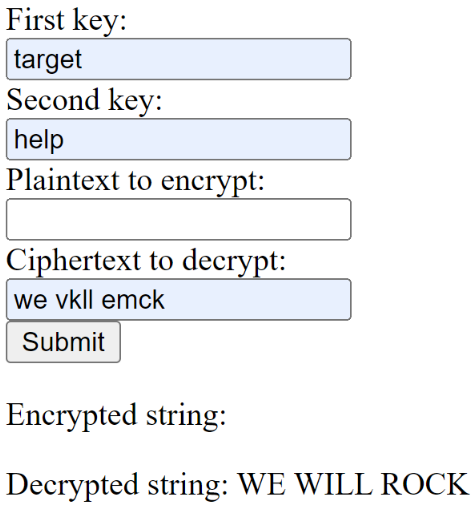
Second key: help

Plaintext to encrypt: we will rock

Ciphertext to decrypt:

Encrypted string: WE VKLL EMCK

Decrypted string:



First key: target

Second key: help

Plaintext to encrypt:

Ciphertext to decrypt: we vkll emck

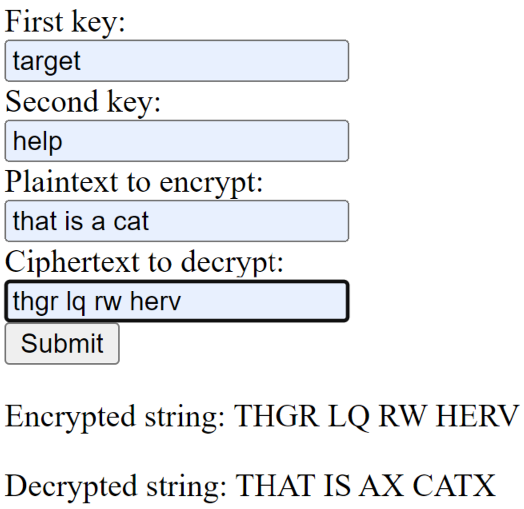
Encrypted string:

Decrypted string: WE WILL ROCK

### The Downside of the Two Square Playfair Cipher:

The bothersome part of the project was dealing with all the exceptions possible. However, we still didn't reach a perfect outcome. Due to adding necessary "X" s and switching "J" s into "I" s, it is not always guaranteed to obtain a perfect decryption. The decryption example provided on page 8 is perfect. The following examples show when decryption isn't flawless:

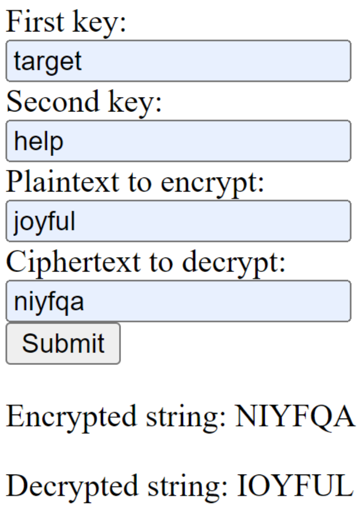
Example 1: "X" inaccuracy


First key:
target
Second key:
help
Plaintext to encrypt:
that is a cat
Ciphertext to decrypt:
thgr lq rw herv

Encrypted string: THGR LQ RW HERV

Decrypted string: THAT IS AX CATX

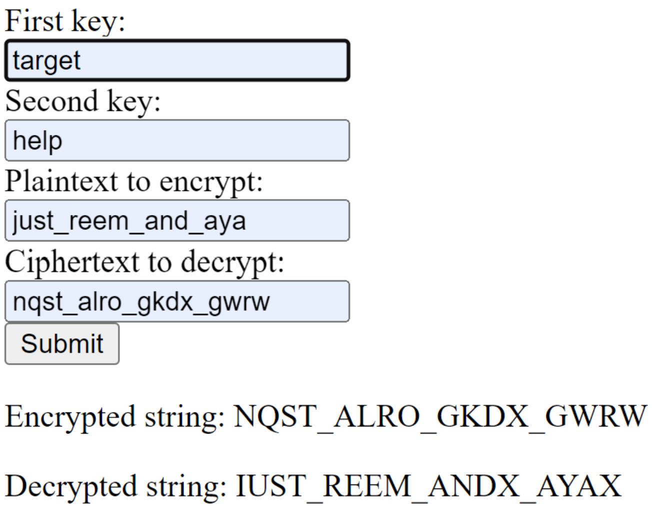
Example 2: "J" inaccuracy


First key:
target
Second key:
help
Plaintext to encrypt:
joyful
Ciphertext to decrypt:
niyfqa

Encrypted string: NIYFQA

Decrypted string: IOYFUL

Example 3: "X" and "J" inaccuracy


First key:
target
Second key:
help
Plaintext to encrypt:
just_reem_and_aya
Ciphertext to decrypt:
nqst_alro_gkdx_gwrw

Encrypted string: NQST_ALRO_GKDX_GWRW

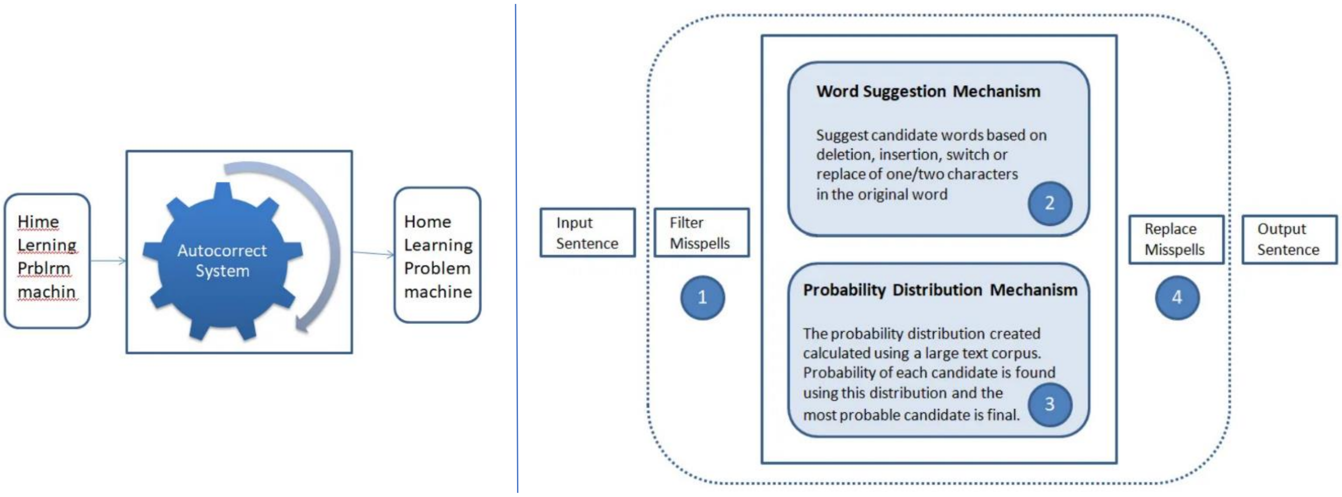
Decrypted string: IUST_REEM_ANDX_AYAX

### Machine Learning and Cryptography:

So, how do we deal with this inconsistency? The traditional solution would be human intervention where a person would decrypt the received cipher text and guess the intended meaning. Nonetheless, this generates human errors. Nowadays, machine learning provides improved security, faster processing, and lower human errors.

In our case, machine learning can serve as an auto-corrector similar to the widely used one in chat and writing applications. We train a machine learning model on a dataset of plaintext-ciphertext pairs using algorithms that learn patterns and associations. Once trained, we can use the model to detect and correct inaccuracies in the decryption of cipher texts.

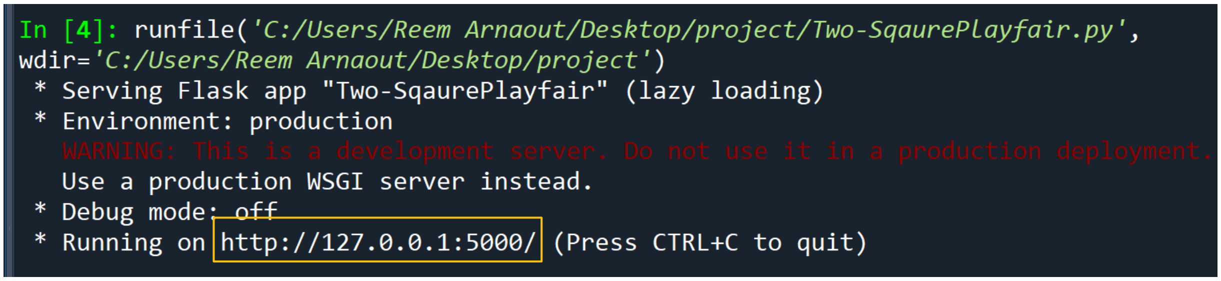
For example, we train the model on the pairs (ENEMY IS NEAR, GOROXY LQ KAAR) even though the decryption of "GOROXY LQ KAAR" is "ENEMYX IS NEAR". When the model encounters "GOROXY", it recognizes that it is "ENEMY" rather than "ENEMYX". This is a mediocre example of how machine learning works since this goes beyond our scope.


Hime
Lerning
Prblrm
machin
Autocorrect System
Home
Learning
Problem
machine
Input Sentence
Filter Misspells
1
Word Suggestion Mechanism
Suggest candidate words based on deletion, insertion, switch or replace of one/two characters in the original word
2
Probability Distribution Mechanism
The probability distribution created calculated using a large text corpus. Probability of each candidate is found using this distribution and the most probable candidate is final.
3
Replace Misspells
4
Output Sentence

Source: "A statistical auto-correct system from scratch" by Pashupati Gupta from "Medium" https://pashupatigupta1998.medium.com/a-statistical-auto-correct-system-from-scratch-e891ddfe141

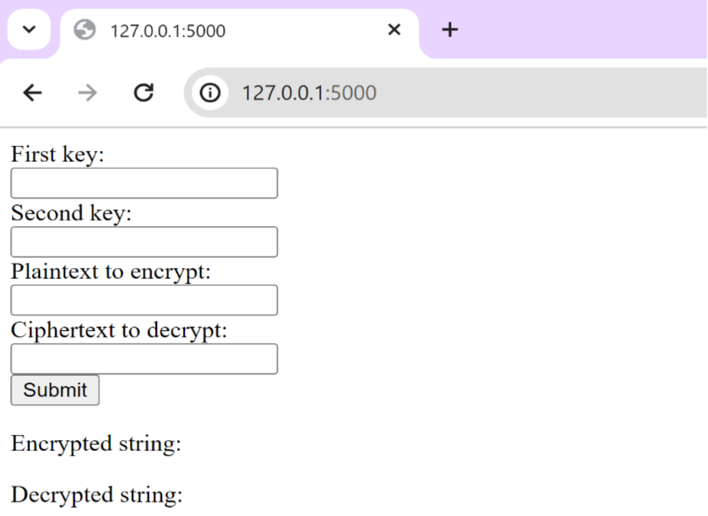
## ○ How to Run the code and HTTP Webpage:

1. After running the code normally, you will receive the following statement:



```
In [4]: runfile('C:/Users/Reem Arnaout/Desktop/project/Two-SqaurePlayfair.py',
wdir='C:/Users/Reem Arnaout/Desktop/project')
 * Serving Flask app "Two-SqaurePlayfair" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

2. Copy the URL link highlighted above. Paste it into a browser of your choice.



First key:

Second key:

Plaintext to encrypt:

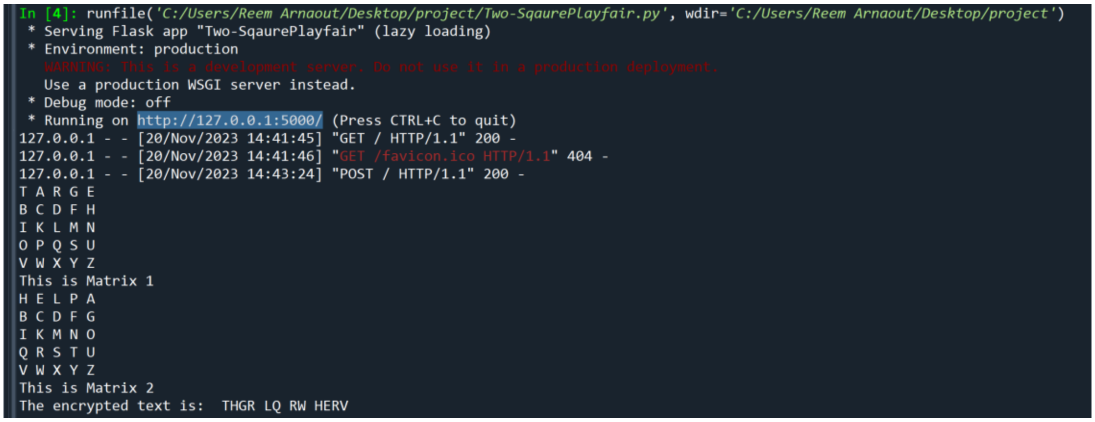
Ciphertext to decrypt:

Submit

Encrypted string:

Decrypted string:

3. Enter your keywords and texts, then press submit to receive the encryption or decryption.
4. On the console where you ran the code, you should receive the following representation (plain text chosen here is “that is a cat”):



```
In [4]: runfile('C:/Users/Reem Arnaout/Desktop/project/Two-SqaurePlayfair.py', wdir='C:/Users/Reem Arnaout/Desktop/project')
 * Serving Flask app "Two-SqaurePlayfair" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [20/Nov/2023 14:41:45] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [20/Nov/2023 14:41:46] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [20/Nov/2023 14:43:24] "POST / HTTP/1.1" 200 -
T A R G E
B C D F H
I K L M N
O P Q S U
V W X Y Z
This is Matrix 1
H E L P A
B C D F G
I K M N O
Q R S T U
V W X Y Z
This is Matrix 2
The encrypted text is:  THGR LQ RW HERV
```

- Appendix:

  ***Python code with HTTP webpage:***

```
"""
Created on Sat Oct 21 16:41:48 2023

@author: Aya Al Baba & Reem Arnaout
"""
from flask import Flask,request, render_template
app = Flask(__name__)

def generate_matrix(keyword):
    keyword=keyword.upper()
    for i in range(len(keyword)):
        if keyword[i]=='J':
            keyword=keyword[:i]+'I'+keyword[i+1:]
    M=[[0 for i in range(5)] for j in range(5)]
    D={}
    k=0
    for i in range(5):
        for j in range(5):
            while k<len(keyword) and keyword[k] in D:
                k+=1
            if k>=len(keyword): break
            M[i][j]=keyword[k]
            D[keyword[k]]=1
            k+=1
        if k>=len(keyword): break
    letters='ABCDEFGHIKLMNOPQRSTUVWXYZ'
    k=0
    for i in range(5):
        for j in range(5):
                continue
    for i in range(5):
        for j in range(5):
            if M[i][j]!=0:
                continue
            while k<len(letters) and letters[k] in D:
                k+=1
            if k>=len(letters): break
            M[i][j]=letters[k]
            D[letters[k]]=1
            k+=1
        if k>=len(letters): break
    return M
```

```
#################################################################


def search_target(matrix,target):
    for i in range(len(matrix)):
        for j in range(len(matrix[0])):
            if matrix[i][j] == target:
                return i, j
    return None


@app.route("/", methods=['POST','GET'])
def main():
    plainstr="
    cipherstr="
    #Generating the Playfair 5x5 two-square matrices
    letters='ABCDEFGHIJKLMNOPQRSTUVWXYZ'
    lettersall='abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'
    keyword1=request.form.get('key1')
    keyword2=request.form.get('key2')
    if keyword1 is not None and keyword1!=" and keyword2 is not None and
keyword2!=":
        for i in range(len(keyword1)):
            if keyword1[i] not in lettersall:
                print("INVALID:The keyword has to be made of letters only!")
                exit()
        matrix1=generate_matrix(keyword1)
        for row in matrix1:
            for element in row:
                print(element, end=' ')
            print()
        print("This is Matrix 1")

        for i in range(len(keyword2)):
            if keyword2[i] not in lettersall:
                print("INVALID:The keyword has to be made of letters only!")
                exit()
        matrix2=generate_matrix(keyword2)
        for row in matrix2:
            for element in row:
                print(element, end=' ')
            print()
        print ("This is Matrix 2")
        #Encryption Process
        plain=request.form.get('encrypt')
        if plain is not None and plain!=":
```

```
    plain=plain.upper()

    k=0
    for i in range(len(plain)):
        if plain[i]=='J':
            plain=plain[:i]+'I'+plain[i+1:]
        if plain[i] not in letters:
            k+=1

    cipher=[]
    i=0
    while i<len(plain):
            if (i==(len(plain)-1)):
                if (((len(plain)-k)%2)!=0) and (plain[i] in letters):
                    plain=plain+'X'
                if (plain[i] not in letters):
                    cipher.append(plain[i])
                    break
            x=search_target(matrix1,plain[i])
            y=search_target(matrix2,plain[i+1])
            if x==None and y!=None:
                cipher.append(plain[i])
                i+=1
                continue
            if x==None and y==None:
                cipher.append(plain[i])
                cipher.append(plain[i+1])
                i+=2
                continue
            if x!=None and y==None:
                plain=plain[:i+1]+'X'+plain[i+1:]
                continue
            if x[1]==y[1]:
                cipher.append(matrix1[x[0]][x[1]])
                cipher.append(matrix2[y[0]][y[1]])
            if x[1]!=y[1]:
                cipher.append(matrix1[x[0]][y[1]])
                cipher.append(matrix2[y[0]][x[1]])
            i+=2

    cipherstr=''.join(map(str,cipher))
    print("The encrypted text is: ", cipherstr)
cipher=request.form.get('decrypt')
if cipher is not None and cipher!='':
    #Decryption Process (same method as Encryption)
    cipher=cipher.upper()
```

```
      k=0
      for i in range(len(cipher)):
        if cipher[i]=='J':
            cipher=cipher[:i]+'I'+cipher[i+1:]
        if cipher[i] not in letters:
            k+=1
      plain=[]
      i=0
      while i<len(cipher):
          if (i==(len(cipher)-1)):
            if (((len(cipher)-k)%2)!=0) and (cipher[i] in letters):
              cipher=cipher+'X'
            if (cipher[i] not in letters):
              plain.append(cipher[i])
              break
          x=search_target(matrix1,cipher[i])
          y=search_target(matrix2,cipher[i+1])
          if x==None and y!=None:
            plain.append(cipher[i])
            i+=1
            continue
          if x==None and y==None:
            plain.append(cipher[i])
            plain.append(cipher[i+1])
            i+=2
            continue
          if x!=None and y==None:
            cipher=cipher[:i+1]+'X'+cipher[i+1:]
            continue
          if x[1]==y[1]:
            plain.append(matrix1[x[0]][x[1]])
            plain.append(matrix2[y[0]][y[1]])
          if x[1]!=y[1]:
            plain.append(matrix1[x[0]][y[1]])
            plain.append(matrix2[y[0]][x[1]])
          i+=2
      plainstr=''.join(map(str,plain))
      print("The decrypted text is: ", plainstr)
  return render_template("index.html",encstr=cipherstr,decstr=plainstr)


if __name__ == "__main__":
  app.run()
```