Playfair Cipher in Python

Aug 21, 2023 • 6 min read
Tags: Cryptography, Python

Playfair cipher is a type of polygraphic cipher that uses 5×5 grid of letters to encrypt and decrypt messages. It was invented by Sir Charles Wheatstone but it is named after Lord Playfair who promoted its use. The Playfair cipher was widely used in the WW1 and WW2 for military communication.

How the Playfair cipher works?

To encrypt and decrypt messages with Playfair cipher, you need the key which is a 5×5 grid of letters. Then you need to split the message into pairs of letters (digraphs) and replace the characters based on some specific rules.

Key generation

Start with a keyword or phrase, for example: “SECRETKEY”. Write the keyword from the top left of the 5×5 grid, and fill the rest of the grid with the remaining letters of the alphabet in order. The grid will look like below:

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

In the Playfair cipher, ‘I’ and ‘J’ are usually treated as the same letter and occupy the same cell in the Playfair square. When encrypting or decrypting, you can interchangeably use ‘I’ and ‘J’ in your plaintext and ciphertext.

Encryption

To encrypt a message, split the message into pairs of letters (digraphs). If both letters are the same or only one letter is left, then add “X” after the first letter. For example, lets encrypt the message “WELL DONE MY FRIENDS”. We split them into: “WE LX LD ON EM YF RI EN DS”.

For each pair of letters, perform the substitution using the following rules:

1. If the letters are in the same row, replace them with the letters to their right (wrap around if necessary).
2. If the letters are in the same column, replace them with the letters below them (wrap around if necessary).
3. If the letters are in different rows and columns, replace them with the letters at the opposite corners of the rectangle formed by them.

We apply the rules above for our message with the 5×5 matrix:

1. WE forms a rectangle, replace it with VC.
2. LX forms a rectangle, replace it with IZ.
3. LD is in a column, replace it with QL.
4. ON is in a row, replace it with PO.
5. EM forms a rectangle, replace it with SN.
6. YF forms a rectangle, replace it with KG.
7. RI is in a column, replace it with BP.
8. EN is in a column, replace it with YV.
9. DS forms a rectangle, replace it with KT.

Our final encrypted message is “VCIZQLPOSNKGBPYVKT”.

Decryption

To decrypt a message, you will need the same 5×5 grid of letters. For each digraph in the ciphertext:

1. If the letters are in the same row, replace them with the letters to their right (wrap around if necessary).
2. If the letters are in the same column, replace them with the letters above them (wrap around if necessary).
3. If the letters are in different rows and columns, replace them with the letters at the opposite corners of the rectangle formed by them.

Lets try to decrypt “VCIZQLPOSNKGBPYVKT”. We split them into digraphs “VC IZ QL PO SN KG BP YV KT” and use the previous 5×5 grid to perform the substitutions:

1. VC forms a rectangle, replace it with WE.
2. IZ forms a rectangle, replace it with LX.
3. QL is in a column, replace it with LD.
4. PO is in a row, replace it with ON.
5. SN forms a rectangle, replace it with EM.
6. KG forms a rectangle, replace it with YF.

7. BP is in a column, replace it with RI.
8. YV is in a column, replace it with EN.
9. KT forms a rectangle, replace it with DS.

Combining all of the decrypted digraphs resulting in “WELXLDONEMYFRIENDS”. We can remove the “X” within the two L’s and we got our original message.

Implementing Playfair cipher in Python

Lets start with the function to generate the Playfair square for a given phrase:

```
def create_playfair_square(phrase):
    key = key.replace('J', 'I').upper() + 'ABCDEFGHIKLMNOPQRSTUVWXYZ'
    key = "".join(dict.fromkeys(key))  # Remove duplicate letters
    grid = [[k for k in key[i:i+5]] for i in range(0, 25, 5)]
    return grid


playfair_square = create_playfair_square('SECRETKEY')
# Output:
# [['S', 'E', 'C', 'R', 'T'],
#  ['K', 'Y', 'A', 'B', 'D'],
#  ['F', 'G', 'H', 'I', 'L'],
#  ['M', 'N', 'O', 'P', 'Q'],
#  ['U', 'V', 'W', 'X', 'Z']]
```

The `create_playfair_square` function accepts a single argument: `phrase`. This phrase will be inserted into the 5×5 grid and the remaining cells will be inserted with the rest of the alphabets.

We also define a helper function to get the coordinates of a character from this Playfair square:

```
def find_location(grid, char):
    """Helper function to get the row and column of the given char"""
    for i in range(0, 5):
        for j in range(0, 5):
            if grid[i][j] == char:
                return i, j
```

Now that we already have the Playfair square and some helper function, we can write the function to encrypt messages. Here is the function:

```
def playfair_encrypt(message: str, key: str) -> str:
    playfair_square = create_playfair_square(key)
    ciphertext = ''

    # Remove non-alphabetic characters
    message = "".join(filter(str.isalpha, message))

    # Handle repeating letters by inserting 'X' between them
    i = 0
    while i < len(message) - 1:
        if message[i] == message[i+1]:
            message = message[:i+1] + 'X' + message[i+1:]
        i += 1

    # Append 'X' if the last block only contain a single character
    if len(message) % 2 == 1:
        message += 'X'

    # For each digraphs, substitute the characters using the grid
    for i in range(0, len(message), 2):
        digraph = message[i:i+2]
        row1, col1 = find_location(playfair_square, digraph[0])
        row2, col2 = find_location(playfair_square, digraph[1])
        if row1 == row2:
            sub1 = playfair_square[row1][(col1 + 1) % 5]
            sub2 = playfair_square[row2][(col2 + 1 ) % 5]
        elif col1 == col2:
            sub1 = playfair_square[(row1 + 1) % 5][col1]
            sub2 = playfair_square[(row2 + 1) % 5][col2]
        else:
            sub1 = playfair_square[row1][col2]
            sub2 = playfair_square[row2][col1]

        ciphertext += sub1 + sub2

    return ciphertext
```

The function `playfair_encrypt` takes two parameters: `message` and `key`, where `message` is the plaintext to be encrypted and `key` is the phrase to generate the Fairplay square.

First, the code will generate the Fairplay square using the provided key. After that, it will pre-process the message to insert ‘X’ between repeating letters. It will also append ‘X’ at the end of the message if the last block only contain a single character.

Inside the loop, each digraph will be replaced with another digraph using the rules explain in the previous section.

The decryption function is similar with the encryption above, we just don't need to pre-process the input message. Instead, we remove the 'X' letters between two similar letters and at the end of the message.

```
def playfair_decrypt(ciphertext: str, key: str) -> str:
    playfair_square = create_playfair_square(key)
    message = ''

    # For each digraphs, substitute the characters using the grid
    for i in range(0, len(ciphertext), 2):
        digraph = ciphertext[i:i+2]
        row1, col1 = find_location(playfair_square, digraph[0])
        row2, col2 = find_location(playfair_square, digraph[1])
        if row1 == row2:
            sub1 = playfair_square[row1][(col1 - 1) % 5]
            sub2 = playfair_square[row2][(col2 - 1 ) % 5]
        elif col1 == col2:
            sub1 = playfair_square[(row1 - 1) % 5][col1]
            sub2 = playfair_square[(row2 - 1) % 5][col2]
        else:
            sub1 = playfair_square[row1][col2]
            sub2 = playfair_square[row2][col1]

        message += sub1 + sub2

    # Remove the 'X' between two similar letters
    i = 0
    while i < len(message) - 2:
        if message[i] == message[i+2] and message[i+1] == 'X':
            message = message[:i+1] + message[i+2:]
        i += 1

    # Remove the last 'X'
    if message[-1] == 'X':
        message = message[:-1]

    return message
```

Putting it all together

Here is the complete code to encrypt/decrypt messages with the Playfair cipher:

```
def create_playfair_square(phrase):
    """
    Generate the Playfair square for the given phrase.
    """
    key = key.replace('J', 'I').upper() + 'ABCDEFGHIKLMNOPQRSTUVWXYZ'
    key = "".join(dict.fromkeys(key))  # Remove duplicate letters
```

```
    grid = [[k for k in key[i:i+5]] for i in range(0, 25, 5)]
    return grid


def find_location(grid, char):
    """
    Helper function to get the row and column of the given char.
    """
    for i in range(0, 5):
        for j in range(0, 5):
            if grid[i][j] == char:
                return i, j


def playfair_encrypt(message: str, key: str) -> str:
    """
    Encrypt a message using the Playfair cipher.
    """
    playfair_square = create_playfair_square(key)
    ciphertext = ''

    # Remove non-alphabetic characters
    message = "".join(filter(str.isalpha, message))

    # Handle repeating letters by inserting 'X' between them
    i = 0
    while i < len(message) - 1:
        if message[i] == message[i+1]:
            message = message[:i+1] + 'X' + message[i+1:]
        i += 1

    # Append 'X' if the last block only contain a single character
    if len(message) % 2 == 1:
        message += 'X'

    # For each digraphs, substitute the characters using the grid
    for i in range(0, len(message), 2):
        digraph = message[i:i+2]
        row1, col1 = find_location(playfair_square, digraph[0])
        row2, col2 = find_location(playfair_square, digraph[1])
        if row1 == row2:
            sub1 = playfair_square[row1][(col1 + 1) % 5]
            sub2 = playfair_square[row2][(col2 + 1 ) % 5]
        elif col1 == col2:
            sub1 = playfair_square[(row1 + 1) % 5][col1]
            sub2 = playfair_square[(row2 + 1) % 5][col2]
        else:
            sub1 = playfair_square[row1][col2]
            sub2 = playfair_square[row2][col1]

        ciphertext += sub1 + sub2

    return ciphertext
```

```
def playfair_decrypt(ciphertext: str, key: str) -> str:
    """
    Decrypt a message using the Playfair cipher.
    """
    playfair_square = create_playfair_square(key)
    message = ''

    # For each digraphs, substitute the characters using the grid
    for i in range(0, len(ciphertext), 2):
        digraph = ciphertext[i:i+2]
        row1, col1 = find_location(playfair_square, digraph[0])
        row2, col2 = find_location(playfair_square, digraph[1])
        if row1 == row2:
            sub1 = playfair_square[row1][(col1 - 1) % 5]
            sub2 = playfair_square[row2][(col2 - 1 ) % 5]
        elif col1 == col2:
            sub1 = playfair_square[(row1 - 1) % 5][col1]
            sub2 = playfair_square[(row2 - 1) % 5][col2]
        else:
            sub1 = playfair_square[row1][col2]
            sub2 = playfair_square[row2][col1]

        message += sub1 + sub2

    # Remove the 'X' between two similar letters
    i = 0
    while i < len(message) - 2:
        if message[i] == message[i+2] and message[i+1] == 'X':
            message = message[:i+1] + message[i+2:]
        i += 1

    # Remove the last 'X'
    if message[-1] == 'X':
        message = message[:-1]

    return message


# Usage example
keyword = 'SECRETKEY'
message = 'THE HACKING BLOG ROCKS'

encrypted = playfair_encrypt(message, keyword)
print(encrypted)   # Will print: CLCGHABFVNDINHCPSARU

decrypted = playfair_decrypt(encrypted, keyword)
print(decrypted)   # Will print: THEHACKINGBLOGROCKS
```

Key Takeaways

To conclude this blog post, let's review the main points we have covered:

1. Playfair cipher is a type of polygraphic encryption technique that uses a 5×5 grid of letters to encrypt and decrypt messages.
2. To generate the Playfair key, you need to choose a keyword or phrase that does not contain any repeated letters, and fill the rest of the grid with the remaining letters of the alphabet in order.
3. You need to split your message into pairs of letters (digraphs), and encrypt or decrypt each pair according to the rules based on their position in the grid.
4. We have shown you how to write python code to encrypt and decrypt text with Playfair cipher in python.