Project 2:
The ADFGVX Cipher: Cryptography of WWI

CSCI 360

March 7, 2017

1 The Main Idea

The **ADFGVX Cipher** was a cipher used during WWI by the German army on the Western front. It combines the ideas of substitution ciphers and transposition ciphers. It was invented by Fritz Nebel in 1918.

This cipher works in two steps. First, it uses a **key square** to encrypt letters. This key square is a 6×6 square which provides instructions for how to perform substitutions for 26 letters and 10 digits. Next it uses a keyword to perform transposition.

	A	D	F	G	V	X
A	**Q**	**C**	**3**	**T**	**6**	**W**
D	**M**	**O**	**E**	**H**	**N**	**L**
F	**8**	**A**	**4**	**2**	**1**	**I**
G	**G**	**B**	**5**	**Z**	**R**	**7**
V	**S**	**X**	**9**	**V**	**U**	**P**
X	**0**	**K**	**J**	**F**	**D**	**Y**

For instance suppose we wih to encrypt the message `HELLO WORLD 123`. Replace each letter/number in this plaintext message with the row-column pair representing the position of that symbol in the above grid. For instance, the given plaintext becomes:

```
DG DF DX DX DD AX DD GV DX XV FV FG AF
```

Note that thus far this is just a substitution cipher. We could use the same frequency analysis we used in our first project to crack this code! So, we need to add another layer of encryption. The second part of the key is to pick a keyword, like in Vigenere's cipher. Let's say our keyword is `LUCKY`. For reasons that will soon be apparent, **the keyword cannot contain double letters**!

First, write the keyword at the top. Take the ciphertext obtained during the first step and write it out in rows beneath the keyword, creating a large column.

```
LUCKY
DGDFD
XDXDD
AXDDG
VDXXV
FVFGA
FXXXX
```

Fill any remaining space with `X`.

Next, rearrange the columns so that the letters of the keyword are in alphabetical order.

```
CKLUY
DFDGD
XDXDD
DDAXG
XXVDV
FGFVA
XXFXX
```

The final ciphertext is obtained by reading the columns top to bottom, left to right. (Do not include the keyword.)

```
DXDXFXFDDXGXDXAVFFGDXDVXDDGVAX
```

2 Instructions

At the end of this project I will expect you to turn in a `.zip` file containing the following:

- A `readme.txt` file containing the first and last names of every member of your group as well as your group number.
- All of your code. You will be graded on whether or not it executes with the correct outputs.

Your file should be submitted to me with the title `project2_group#` where `#` denotes your group number.

3 Encryption Using ADFGVX

The goal of this project is to write a program which can encrypt a phrase using the ADFGVX cipher.

3.1 Challenge 1: Getting A Keyword

The first function you write will be called `KeyGen`. The important concept this function must perform is verifying that the chosen keyword contains no double letters. So, `HELLO` and `111` are invalid keywords, but `PIKACHU` and `ORANGE` are valid. It may have the following psuedocode:

```
INPUT: None
OUTPUT: keyword
*******************
def KeyGen():
    while True:
        Ask user for keyword

        Check if keyword has any repeated letters

        If the keyword has no repeated letters:
            return the keyword
```

The above psuedocode is modeled like a Python program, but you can use any programming language of your choice. Also note that you are not required to follow this psuedocode if you think of a different way you would like to model this program – there are many different ways you could write this function!

Hint: The most difficult portion of the above code will be coming up with a way to check if the keyword has any repeated letters; experiment with different techniques until you find one which works. You might want to use Python's `count` method. For instance, if `a = 'foo'` then `a.count('f')` returns 1, while `a.count('o')` returns 2. You want a keyword where the count of each letter in the keyword is 1.

3.2 Challenge 2: Encryption Part 1, Substitution

The next function you write will be called `Encrypt1`. It's input will be a plaintext message `plaintext` and its output will be the ciphertext after substitution via the chart above. I recommend we first rewrite the chart as follows. Below, the plaintext letter is on the left, and its corresponding row-column value is on the right.

A	FD
B	GD
C	AD
D	XV
E	DF
F	XG
et cetera	

In the .zip file for this project I have included a C and a Python representation of the chart for your convenience – you may copy/paste it into your code. It is in the .txt file titled `keysquare.txt`. I recommend storing this at the top of your code as a global variable named `keysquare`.

Your job for the `Encrypt1` function is to use substitution to create a ciphertext where each letter in the plaintext is represented by its value in the keysquare. Ignore all spaces and punctuation. For instance, the input `*?H E L L O WORLD 123!!!~#` would return `DGDFDXDXDDAXDDGVDXXVFVFGFA`.

Python psuedocode for Challenge 2:

```
INPUT: plaintext
OUTPUT: ciphertext
****************************

def Encrypt1(plaintext):
    Initialize the ciphertext to an empty string.

    for letter in plaintext:
        If the letter is represented in the keysquare (A-Z, 0-9):
            Append its corresponding keysquare value to the ciphertext

    Return the ciphertext
```

Again this psuedocode is only a suggestion. You may write your function differently if you wish to.

3.3 Challenge 3: Apply the Keyword

Next you write a function to split up the ciphertext into columns corresponding to the keyword. Recall that for plaintext `HELLO WORLD 123` and keyword `LUCKY`, we split the ciphertext returned by the function in Challenge 2 as follows:

```
LUCKY
DGDFD
XDXDD
AXDDG
VDXXV
FVFGF
AXXXX
```

I recommend storing this as a list of lists (Python) or an array of arrays (C). For example,

```
ciphertext_list = [ ['D', 'G', 'D', 'F', 'D'],
                    ['X', 'D', 'X', 'D', 'D'],
                    ['A', 'X', 'D', 'D', G'],
                    ['V', 'D', 'X', 'X', 'V'],
                    ['F', 'V', 'F', 'G', 'F'],
                    ['A', 'X', 'X', 'X', 'X'] ]
```

Items in this double list can be accessed by `ciphertext_list[row][column]`. For example `ciphertext_list[5][0] = 'A'`.

Observe that in order to set this up, we initialize `ciphertext_list` first. Each sub-list contains the same number of letters as there are in the keyword, in the order that they appear in the ciphertext. Thus we must iterate over both the length of the keyword as well as the letters in the ciphertext.

We will formulate this by iterating over the letters in the keyword, and maintaining a separate counter which iterates modulo the length of the keyword.

Python psuedocode for Challenge 3 is as follows. Note that there are many ways you could implement this; if you are familiar with more advanced Python techniques such as list comprehensions, this code can be shortened to only a handful of lines. The following is set up to use only introductory Python techniques.

We first wish to convert the ciphertext to a list. For instance, the ciphertext `ciphertext = 'VFAQ'` will be converted to `c = ['V', 'F', 'A', 'Q']`. Then we will use **list slicing** to split this into sublists. For instance, split the previous example into sublists of length 2:

```
ciphertext = [ ['V', 'F'], ['A', 'Q'] ]
```

or into sublists of length 3:

```
ciphertext = [ ['V', 'F', 'A'], ['Q', 'X', 'X'] ]
```

Note that we had to pad the second list with `'X'`'s.

```
INPUT: ciphertext (string) and the length of the keyword (integer)
OUTPUT: ciphertext_list, a list of lists (or array of arrays in C)
*****************************************
def Encrypt2_Setup(ciphertext, L):
    Convert ciphertext to a list using the list() function, store as c

    Initialize ciphertext_list to be an empty list

    for i in range(0, len(c), L):
        ciphertext_list = c[i: i+L]

    Verify that the last sub-list has the correct number of entries.
    If it does not, pad it with the appropriate number of 'X' entries.

    Return the ciphertext list
```

I included code for the most difficult part of the above function – namely, splitting the list into length-L sublists. You just need to fill in the rest. The most difficult part left to you is to verify that the last sub-list has the correct number of entries, and if not, to pad it with `'X'` the appropriate number of times.

3.4 Challenge 4: Encryption Part 2, Transposition

Now we wish to apply the transposition to the `ciphertext_list` obtained in the previous function. First we must find the transposition, then apply that transposition to each row in the ciphertext list.

I have provided functions which you may use for this part of the program as `challenge4.py` in the zip file. You are not required to do anything for challenge four other than copy and paste the functions into your code, and set them up so that they work with your code. You may rewrite these functions in a different way if you wish to.

I will walk you through this code. First there is a function called

```
Encrypt2_Transpose(ciphertext_list, keyword)
```

This function takes as input the ciphertext list returned from Challenge 3, as well as the keyword. Let's say we have the ciphertext list

```
ciphertext_list = [['D', 'G', 'D', 'F', 'D'],
                   ['X', 'D', 'X', 'D', 'D'],
                   ['A', 'X', 'D', 'D', 'G'],
                   ['V', 'D', 'X', 'X', 'V'],
                   ['F', 'V', 'F', 'G', 'A'],
                   ['F', 'X', 'X', 'X', 'X']]
```

and the keyword `LUCKY`. The function begins with the following:

```
temp = list(keyword)
temp.sort()
```

This takes the keyword and turns it into a sorted list, stored as `temp`. After running that code on the keyword `LUCKY` we would be left with

```
temp = ['C', 'K', 'L', 'U', 'Y']
```

Next the function executes the following:

```
permutation = []
for letter in keyword:
    n = temp.index(letter)
    permutation.append(n)
```

Using the previous example would yield `permutation = [2, 3, 0, 1, 4]`.

Next the function executes the following:

```
ciphertext_list = Permute(ciphertext_list, permutation)
```

This calls a function `Permute` which I will not describe – you may read over it on your own if you like. This function re-orders the values in `ciphertext_list` according to the given permutation. After this line, the value of the ciphertext list is reassigned to the following:

```
ciphertext_list = [['D', 'F', 'D', 'G', 'D'],
                   ['X', 'D', 'X', 'D', 'D'],
                   ['D', 'D', 'A', 'X', 'G'],
                   ['X', 'X', 'V', 'D', 'V'],
                   ['F', 'G', 'F', 'V', 'A'],
                   ['X', 'X', 'F', 'X', 'X']]
```

The remaining portion of this code reads through this column-by-column and returns this as the ciphertext.

3.5 Challenge 5: Write a main function

Now write a main function which does the following:

1. Get the keyword from the user by calling `KeyGen`
2. Get the plaintext from the user
3. Call `Encrypt1` to get the first layer of ciphertext
4. Call `Encrypt2_Setup`, which sets up the ciphertext for the final layer of security.
5. Call `Encrypt2_Arrange`, which returns the final ciphertext
6. Print the ciphertext.

An example run would look like follows:

```
What is your keyword? LUCKY
What is your plaintext? HELLO WORLD 123!!!
Your ciphertext is:
DXDXFXFDDXGXDXAVFAGDXDVXDDGVFX
```

Your project will be graded largely on whether you are able to call these functions correctly and output the correct ciphertext.

3.6 Bonus Challenge: Decryption

The bonus challenge is to write a function called `decrypt` which takes the ciphertext and keyword as input and returns the ciphertext. Completing this challenge gives you a free `100%` on your lowest (not dropped) quiz score and 3 bonus points on your first midterm exam.