**Fall 2025 CIS 3362 Homework #7: Elliptic Curve Challenge Solution**

1) (50 pts) In this question, you will decode a ciphertext that will reveal Alice's Private Key which will be necessary to solve question 2. Alice's Private Key is a large integer.

The following ciphertext below was created via the ADFGVX cipher using the keyword "ELLIPTICURVE" and the following 6 x 6 grid:

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

```
VXGVVDVDGVGVVGFDGFFGXGDXVGVVVGDFVXVGGVFFGDGVFVGGVXXVVGXDGGVG
DXGGDFVXVGVGDFVFVXXFDXVXGGVDFGXVVFVFGGDVXGVGGGXDDGFVXFFGVGFF
GFGXVVXXGXGFGVXFVVGVVDGDFVVDGDVXGVVDGX
```

Please feel free to do this by hand, or more likely, write a program. You may build off of sample programs from the course website. Also, it's best to copy paste the 6 x 6 grid above so the number 0 and the letter O don't get confused.

**Solution**
Probably too hard to do by hand.

Write a program to decrypt the cipher text. Posted on the course webpage is code that does adfgvx encryption: https://www.cs.ucf.edu/~dmarino/ucf/cis3362/progs/adfgvx.java

It makes sense to add methods to this code for decryption.

The attached file, **adfgvx_decrypt.java**, was written to handle decryption. In summary, to undo the transposition, use the same looping structure, but move the item in the current index back to where it came from before the transposition. To undo the substitution, a direct look up in the 6 by 6 input grid is necessary. Thus, for the code, this 6 by 6 grid was added as an instance variable to the adfgvx object.

When running the decryption on the given information, stored in the attached file h7_q1_todecrypt.txt, we get the following output:

```
D4FDWGPZ9M0GU9F4Z4525DJH7IWD61JDU0GGH9GDWMN4MA6DD7W8691W9D492MW5
DU3G5U4IWGI7FN6
```

Unfortunately, this isn't a number, which is what we were expecting. Looking at the keyword, we notice that it's a misspelling of Elliptic Curve. In particular, there is only one C written in the assignment keyword, but the real word has two C's. It's reasonable to guess that there was a typo (in fact this could have happened accidentally on the battle field in WWI!), so let's try the new keyword of "ELLIPTICCURVE". I created a new input file, h7_q1_todecrypt_v2.txt with this keyword and reran my program. It now produces the output:

**6342379028871672578799006438375742821981052796152446398587360791435451555403625**

Voila!!! We figured it out!!!

The probability we did the wrong work but got all digits is extremely low, it's $(\frac{10}{36})^n$, where n is the number of characters in the decrypted message, so we can be quite confident that we found the typo and this was the intended number to use for the second portion of the assignment!

Summary of attached files for this question (when you unzip H7-Q1-Files.zip):

1. adfgvx_decrypt.java
2. h7_q1_todecrypt.txt
3. h7_q1_todecrypt.out
4. h7_q1_todecrypt_v2.txt
5. h7_q1_todecrypt_v2.out

2) (75 pts) The message to break for this question resides in the file

**`h7_q2_ciphertext.txt`**

The file first lists the Alice's Public Elements for her cryptosystem:

1. The Curve for the system with the prime number (labeled x in the file) as well as the constants a and b (as described in class).

2. The "Generator" point for the system is listed as two separate values gx and gy, for the x and y coordinates, respectively, for the curve.

3. Finally Alice's Public Key is listed, though you won't actually have to use this.

Following this there is the ciphertext. Each pair of points C1 and C2, are listed on consecutive lines, followed by a blank like. Each pair of points encrypts 256 bits or 32 bytes of text. There are a total of 17 blocks of text, so that means the plaintext will be 32 x 17 = 544 total Ascii characters.

You must first solve question 1 to get Alice's Private key, and then use that to decrypt.
There is no prize for question #1, but there is a prize for the first team that solves question #2.

Good luck!!!

**Solution**

I edited my posted ECC code online adding a function called solveh7q2(). All of the code is included in the zip file, **ECC-Solve-H7-Q2.zip.** In this function, I hard-code the given information to me as follows, along with Alice's secret number I figured out in question 1. Here is that portion of the code:

```
# Copy from given file.
x = 548061460792020121498633043842389816126897339705730905851067096571429438809443 39
a = 364462835230875022871151830713899506459920532984062000734860924839105565126319 03
b = 427348161586230871504185589300087927375321286561039395507535987233741920379149 99

# Make the curve and find some random point on it.
myC = EllipticCurve(x, a, b)

# This was the point given in the file.
gx = 218005462632118866488511347541362484663356816162590436831400526418095374931233 16
gy = 756783820271599397022825746114011688298932746966544548759959624348528598370679 3

# This is Alice's private key, that we got from solving question #1.
nA = 634237902887167257879900643837574282198105279615244639858736079143545155540362 5

# Creates the ECC object.
aliceECC = ECC(myC, gx, gy, nA)
```

From here I had some difficulty and I am not sure why I couldn't split a Python string on multiple characters. (Yes, this took me the longest time to fight through, of all things...) Anyway, I ultimately had to write my own cleanint function. I read in the points as was posted online and then did the following:

```
# Read in both points
c1 = input().strip()
c1 = c1.split(",")

c2 = input().strip()
c2 = c2.split(",")
```

But then, c1 and c2 were lists of strings, but those strings had stray characters in them. Here is the cleanint function I wrote to process those strings:

```
# Returns an integer from s where s might have non-digit characters
# at beginning or end.
def cleanint(s):

    t = ""
    for x in s:
        if x >= '0' and x <= '9':
            t = t + x
    return int(t)
```

Finally, could create the necessary point objects as follows:

```
c1Pt = Point(myC, cleanint(c1[0]), cleanint(c1[1]))
c2Pt = Point(myC, cleanint(c2[0]), cleanint(c2[1]))
```

To decrypt, I just did this:

```
msgBack = aliceECC.decryptBlock([c1Pt, c2Pt], 32*8)
print(msgBack)
```

Here is the output I received from the 17 blocks:

```
Okay, this is the big test! Will
 be ECC code actually work??? If
 you are reading this, then the
answer is yes. If you are the fi
rst to read this, then go to a o
ffice near me which has a prime
factorization of the form 5p, wh
ere p is the other prime. The of
fice is maybe 50 feet from mine.
 There's a corkboard. On that bo
ard is a UCF football schedule.
The winning note (an orange post
 it), is right behind that footb
all schedule. Just take the thum
b tack off and you'll find the n
ote behind the schedule. Good lu
ck!----------------------------
```

Cleaning this up a bit, we get

```
Okay, this is the big test! Will be ECC code actually work??? If
you are reading this, then the answer is yes. If you are the first
to read this, then go to a office near me which has a prime
factorization of the form 5p, where p is the other prime. The
office is maybe 50 feet from mine. There's a corkboard. On that
board is a UCF football schedule. The winning note (an orange post
it), is right behind that football schedule. Just take the thumb
tack off and you'll find the note behind the schedule. Good luck!
```

My office number is 240, so the nearest multiples of 5 are 230, 235, 245, 250, 255, these are 5 x 46, 5 x 47, 5 x 49 and 5 x 50. The only number that fits the description is 235 = 5 x 47, since 47 is prime. That office belongs to Dr. Mike Borowczak. I don't even know if he knows I hid the prize on his board!!!

Summary of attached files:

1. H7-Q2-Files.zip

Upzip and there are 5 .py files and one text file. I edited ECC.py only and created h7_q2_adjusted.txt to make reading in the text a bit easier.