[100 Points] Implement the encryption and decryption functions in Python for an Affine Cipher that takes as input the corresponding ciphertext/-plaintext, and arbitrary alphabet A such that $26 \leq |A| \leq 256$. Make sure that the cipher implementation warns users about invalid inputs.

My Python implementation for this is shown in Appendix A.

[40 Points] Consider an unfair coin where the two outcomes, heads and tails have probabilities $p(\boldsymbol{heads}) = p$ and $p(\boldsymbol{tails}) = 1 - p$

(a) If the coin is flipped two times, what are the possible outcomes along with their respective probabilities?

1. HH, $p(HH) = p^2$
2. HT, $p(HT) = p \cdot (1-p)$
3. TH, $P(TH) = (1-p) \cdot p$
4. TT, $p(TT) = (1-p)^2$

(b) Show that the entropy in part (a) is $-2p\log_2(p) - 2(1-p)\log_2(1-p)$. How could this have been predicted without calculating probabilites in part (a)?

One can calculate the entropy using Shannon's Entropy, which is Equation 1 below.

$$H(X) = -\sum_{x \in X} p(x) \cdot \log_2 p(x) \tag{1}$$

For our case, there are four possibilities, with the probabilities shown above.

$$H(X) = -p^2 \cdot \log_2 p^2 - 2 \cdot p \cdot (1-p) \cdot \log_2[p \cdot (1-p)] - (1-p)^2 \cdot \log_2(1-p)^2$$

This can be expanded by using the properties of exponents to simplify into the expected expression, as shown below:

$$H(X) = -2p^2\log_2 p - 2p(1-p)\log_2 p - 2p(1-p)\log_2(1-p) - 2(1-p)^2\log_2(1-p)$$

Now group by $\log_2 p$ and $\log_2(1-p)$:

$$\begin{aligned} &\text{Terms for} \log_2 p: \\ \hline &-2p^2\log_2 p - 2p\log_2 p + 2p^2\log_2 p \\ &= -2p\log_2 p \end{aligned}$$

$$\begin{aligned} &\text{Terms for} \log_2(1-p): \\ \hline &-2p\log_2(1-p) + 2p^2\log_2(1-p) - 2\log_2(1-p) + \ldots \\ &+4p\log_2(1-p) - 2p^2\log_2(1-p) \\ &= (2p-2) \cdot \log_2(1-p) \\ &= -2 \cdot (1-p) \cdot \log_2(1-p) \end{aligned}$$

The sum of the two terms above are the expression for entropy:

$$H(X) = -2p\log_2 p - 2(1-p)\log_2(1-p)$$

This value could be predicted by assuming the worst case entropy: $p(heads) = p(tails)$. This leads to the following expression for entropy: $H(X) = log_2|A|$, where $|A|$ is the number of possible outcomes. For 2 coin flips, the maximum entropy would be $H(X) = log_2 4 = 2$. One can also predict the entropy y multiplying

[60 Points] Decipher the following ciphertext using ciphertext-only cryptanalysis. Note the alphabet used to create is this ciphertext is as following: [abcdefghijklmnopqrstuvwxyz]

[30 Points] Ciphertext #1 (Affine)

```
azwcwlugblyciuohxfoxaiallcsrrwhxobzzupubzxfuewbcxaxsxawbwpxfusbaxu
zcxoxucokoabcxollubugaucpwhuakbobzzwgucxaexfoxaialljuohxhsupoaxfob
zollukaobeuxwxfucoguxfoxaxoquxfacwjlakoxawbphuulyiaxfwsxobygubxolh
ucuhnoxawbwhrshrwcuwpunocawbobzxfoxaialliullobzpoaxfpsllyzacefohkux
fuzsxaucwpxfuwppaeuwbifaefaogojwsxxwubxuhcwfulrguk
```

I decrypted this ciphertext using a brute force attack, determining the key is $\alpha = 21$, $\beta = 14$, producing the below output:

```
C:\Users\Micah Hayden\Documents\AFIT\SP2019\Data Security\Assignments\HW2\PythonScripts>
python HW2_Affine_Cipher.py
Starting decrypt with alpha = 21 and beta = 14:

The output is:
idosolemnlyswearthatiwillsupportanddefendtheconstitutionoftheunitedstatesagainstallenemi
esforeignanddomesticthatiwillbeartruefaithandallegiancetothesamethatitakethisobligationf
reelywithoutanymentalreservationorpurposeofevasionandthatiwillwellandfaithfullydischarge
thedutiesoftheofficeonwhichiamabouttoentersohelpmeg
```

Figure 1: Output of my Affine python script with $\alpha = 21$, $\beta = 14$

[20 Points] Ciphertext #2 (Vigenère)

```
kvqkqdgepdakywcjvzclkokdnkwhrgtlcffvgxgffljwegpkvavmvaqfqxvzgmpavwfkvsvwus
iskfulcdnwpwoagkhgtwkypspvfgowulkuvzclkokdntgstltmgxcavzcffsndgykspuglqljwuso
wcffljsvayandqtgqvzggtvgjughljwrjgkkvgfvghljwwfklgvulclgkcffljwqjfwtkqxvzgghxku
gjusrhqaplgvqngjowcuegtvkfilqjgywdclkgpkcffljwwfkxqjouqvgghekdklcjabwkvaewugj
wnhowigf
```

I utilized the vigenereHacker.py script described in [1] to determine the likely key to the ciphertext, with the output shown below:

```
C:\Users\Micah Hayden\Documents\AFIT\SP2019\Data Security\Assignments\HW2\
PythonScripts>python vigenereHacker.py
Kasiski Examination results say the most likely key lengths are: 2 4 3

Attempting hack with key length 2 (16 possible keys)...
Possible letters for letter 1 of the key: C N I G
Possible letters for letter 2 of the key: S G H R
```

Figure 2: Likely keys of given ciphertext

I then utilized an online vigenere cipher decoder [2], with the first likely key **"cs"**. This produced the following plaintext:

```
idosolemnlyswearthatiwillsupportanddefendtheconstitutionoftheunitedstatesa
gainstallenemiesforeignanddomesticthatiwillbeartruefaithandallegiancetothesam
eandthatiwillobeytheordersofthepresidentoftheunitedstatesandtheordersoftheoffic
ersappointedovermeaccordingtoregulationsandtheuniformcodeofmilitaryjusticesoh
elpmegod
```

[10 Points] Ciphertext#3 (Vigenère)

```
ujltkvbpxowvvcoqcubiubrkjofvtlpwvbuwplxtvkpytvkflnbzqxdcqgkqeqxuykbvlturvpxtw
dmcepwwjvlunrpmvtqrsflocuzcqerobkqujduarvyrvngujlomqpvkjpjxcvxtroizjvkjvlohddv
qpvkjpjjuzsqqhylujlukwjukjikmaxowvvcoujrviktvxealucqevkjpnbdrvvludxuarvuwjnonc
boqwgkqecqzvkjpjpfgcwwjhbpwzptnbvlucejvbpwpjikuhlofrvvwsawpfrtqzjvkpwwbpbdq
ddjloi
```

To crack this ciphertext, I utilized the same sequence and resources [1] and [2].

```
C:\Users\Micah Hayden\Documents\AFIT\SP2019\Data Security\Assignments\HW2\
PythonScripts>python vigenereHacker.py
Kasiski Examination results say the most likely key lengths are: 3 2 4

Attempting hack with key length 3 (64 possible keys)...
Possible letters for letter 1 of the key: B I N G
Possible letters for letter 2 of the key: C J H I
Possible letters for letter 3 of the key: D X I J
```

Figure 3: Likely keys of given ciphertext

I attempted the first likely key, **"bcd"**, which produced the below plaintext:

```
thisisanunusualparagraphimcuriousastojusthowquicklyyoucanfindoutwhatissounusu
alaboutititlookssoordinaryandplainthatyouwouldthinknothingwaswrongwithitinfact
nothingiswrongwithititishighlyunusualthoughstudyitandthinkaboutitbutyoustillma
ynotfindanythingoddbutifyouworkatitabityoumightfindouttrytodosowithoutanyco
aching
```

The "unusual" aspect of the plaintext is that there are no occurrences of the letter "e".

References

[1] Al Sweigart. *Cracking Codes with Python*. No Starch Press, 2018.

[2] Franz Friederes. 2019. URL: https://cryptii.com/pipes/vigenere-cipher.

Appendix A: Python Script for Affine Cipher

```python
import string
import math

def ModularInverse(alpha, alphabet):
  running = True
  size  = len(alphabet)
  inv_alpha = 1
  while (running and inv_alpha < size):
    if (  (alpha * inv_alpha)  % size == 1):
      #print( "Modular inverse of {0} is: {1}".format(alpha, inv_alpha) )
      running = False
      return inv_alpha
    else:
      inv_alpha += 1

  # No modular inverse for alpha
  return 0

def affineDecrypt(inv_alpha, beta, ciphertext, alphabet):
  output = ""
  for letter in ciphertext:
    in_index = alphabet.index(letter)
    out_index = ( inv_alpha * (in_index - beta) ) % len(alphabet)
    output += alphabet[out_index]
  return output

def affineEncrypt(alpha, beta, plaintext, alphabet):
  output = ""
  for letter in plaintext:
    in_index = alphabet.index(letter)
    out_index = ( in_index * alpha + beta) % len(alphabet)
    output += alphabet[out_index]
  return output


def checkInputs(alpha, beta, alphabet):

  # Is alphabet in length requirements:
  if (len(alphabet) < 26 or len(alphabet) > 256):
    print("Invalid alphabet size")
    return 1

  # Is alpha within bounds:
  if (alpha == 0 or alpha > len(alphabet)-1):
    print("Invalid value for alpha")
    return 1

  # Is beta within bounds:
  if (beta < 0 or beta >= len(alphabet)):
    print("Invalid value for beta")
    return 1

  # Does alpha have modular inverse:
```

```python
    gcd = math.gcd(alpha, len(alphabet))
    if (gcd != 1):
        print("Invalid value for alpha - no modular inverse")
        return 1

    return 0

def main():

    # Input your alphabet here:
    alphabet = "abcdefghijklmnopqrstuvwxyz"

    # input your key values here:
    alpha = 21
    beta = 14

    # Which mode are we in:
    encrypt = False

    if (checkInputs(alpha, beta, alphabet)):
        print("Exiting due to invalid input")
        return

    if encrypt:
        print("Starting encrypt with alpha = {0} and beta = {1}: \n".format(alpha, beta))
        plaintext = "firstthesentenceandthentheevidencesaidthequeen"
        output = affineEncrypt(alpha, beta, plaintext, alphabet)
    else:
        print("Starting decrypt with alpha = {0} and beta = {1}: \n".format(alpha, beta))
        ciphertext = "azwcwlugblyciuohxfoxaiallcsrrwhxobzzupubzxfuewbcxaxsxawbwpxfusbaxuzcxox"
        output = affineDecrypt(ModularInverse(alpha, alphabet), beta, ciphertext, alphabet)

    print( "The output is: \n" + output )

if __name__ == "__main__":
    main()
```

PythonScripts/HW2_Affine_Cipher.py