# Homework 2: Crypto Basics

### Problem 1:-

To determine the size of the key, I performed the Kasiski approach. Then I utilize all potential key combinations to deduce the plaintext.

```
substitution_cipher = 'OBRGXIMYAZZAWCATBNMUYYHAZNVGFCXPVVSIJSVLKIFAVGBIECAZSBWGRGRQWUCHMMOCYE\
FLGQQNKFSHQMGYALNKCIJQVEKVWXNFOYFYQBESGOYTXMAYTXSISNBPMSGOJBKFWRUTTMLS\
BNQMLLRGFNZUAWHZLBRVZGHUVZMCKJEHSLSWGXCNZYEXRIMLPXRIXNUXRNSNRPHFDHBMAY\
WKHTKNGNUXRNJUVGMYNYEYNLYYGPGYFSBNQQWUCHMMSLRWTFDYQRNOJUEWNLVUZIDHWXLH\
TLKNEXMALBRQGUMMGXCUFXLHTLLLRTROJYFIDHLIGADLUBVXENSCALVCDFFIQCFAHISILU\
XXZXNUAMZAWISRNOJYKMQYECGRSBWHAHLUFBBPDPWLJBRYOCYEAYSVYXSISPRKSNZYPHMM\
WKHXMWWMGAZNEOFMDHKORMGOKNUHTAZQRAZPWBRTQXGZFMTJAXUTRNWCAPZLUFRODLFYFL\
GUKHRODLTYRGRYWHNLRIUCNMDXOCGAKIFAQXKUQMVGZFDBVLSIJSGADLWCFGNCFMGTMWWI\
STBIMHGKXBSPVGFVWHRYHNWXSKNGHLBENHYYQPZLXUEXNHDSBGDQZIXGNQKNUXCCKUFMQI\
MMRYEYUNFHEUDIAZVUJWNGQYSFVSDNZYFNOLWGRBLJGLGTMWWISKZJAXVMXCFVEBMAAHTB\
SNGUPENMWCGBRIFFLHMYOBBBRNZIEHTAZFLTBKMUVGSYVQVMGNZYROHFKISPZLOBBVZHLB\
BKNOYBYRTHVYELSUFXGADJJISBSUTFRPZSGZPTQLQCAZHNGHGADMCCYEEODARGDLSFQHDM\
FIGKZCKYNLDWGHQEDPQHRBSBWLNKDBAMFNOJDSJTFIFMYHZXWXZHQYLBNGSQAWRHMWWQNK\
HMVYPEZLWXUXVCDFAHSQSMGXOLWWVHTMLCZXHHOUVMHHYZBKQYAHSHQWWGRGSMFIEPHFDB\
RMTLFBVLZLESOTBEXIEYQYKBFNOJDCRLAOLWEHRMWMGADYFYZRRZJIAMHYJQVMGIMNQXKU\
QNUXUUDORHENAGRMGULCFUDCFANEHNLFRTGYSXBYXIMLBIOIFYAMGUKWBNMNWXSHQGGLRM\
GUFYVMGYJHHFDLAWNEROHYEBNLANLHQNZYABBYKNPTKWMFNMHIFMJBSBJYTTQXLIPHLGAM\
FTQCSN'

total_char = 26

# this method will give us population variance of a cipher text.
def population_variance(cipher):
    frequency = [0]*total_char
    for i in range(0,len(cipher)):
        frequency[ord(cipher[i].capitalize())-ord('A')] += 1
    sum_ = 0
    mean = 1/total_char
    for i in range(0,total_char):
        sum_ += (frequency[i]/len(cipher) - mean)**2
    return sum_/total_char

# this method will create chunks of cipher and calculate population variance for every chunk of cipher
def kasiski(cipher, no_of_chunks):
    chunks=[]
    for j in range(0 , no_of_chunks):
        chunks.append("")
    for j in range(0, len(cipher)):
        chunks[j%no_of_chunks] += cipher[j]
    sum_=0
    for c in chunks:
        sum_ += population_variance(c)
    return sum_/no_of_chunks

# This method create plain text for given ciper text and key
def decrypt(cipher, key):
    plain_text = ""
    for i in range(0, len(cipher)):
        plain_text = plain_text + chr((ord(cipher[i])-ord(key[i%len(key)]))%26+ord('A'))
    return plain_text

# this method find the letters with maximum frequencies.
def max_frequency_letter(chipher, select_char):
    frequency=[0]*total_char
    for i in range(0,len(chipher)):
        frequency[ord(chipher[i].capitalize())-ord('A')]+=1
    frequency=sorted(range(len(frequency)), key=lambda x: frequency[x], reverse=True)
    letters = []
    for i in range(0, select_char):
        letters.append(chr((frequency[i]-4)%26+ord('A')))
    return letters

# we call kasishi method repeatedly and check for highest population variance to get the key length
max_variance = 0
key_length = 0
for i in range(1,10):
    temp_variance = kasiski(substitution_cipher, i)
    if(max_variance < temp_variance):
      max_variance = temp_variance
```

```
      key_length = i
# print(max_variance)
# print(key_length)

# This loop will create key_length different caesar ciphers
caesar_ciphers = [""]*key_length
for i in range(0, len(substitution_cipher)):
    caesar_ciphers[i % key_length] += substitution_cipher[i]
# for i in range(0, len(caesar_ciphers)):
#   print(caesar_ciphers[i])

# return all the possible keys made up with letters with maximum frequencies
def possible_keys(select_char):
    letters = []
    for i in range(0, len(caesar_ciphers)):
        letters.append(max_frequency_letter(caesar_ciphers[i], select_char))
    return [(c0+c1+c2+c3+c4+c5) for c0 in letters[0] for c1 in letters[1] for c2 in letters[2]
            for c3 in letters[3] for c4 in letters[4] for c5 in letters[5]]

#Below code trying all the possible generated keys and check for highest population variance in plain text
guessed_keys = possible_keys(3)
predicted_key = ''
predicted_key_variance = 0
predicted_plain_text = ''
for key in guessed_keys:
    temp_plain_text = decrypt(substitution_cipher,key)
    temp_variance = population_variance(temp_plain_text)
    if predicted_key_variance < temp_variance:
      predicted_key = key
      predicted_key_variance = temp_variance
      predicted_plain_text = temp_plain_text

print(predicted_key + "=" + str(predicted_key_variance) + "\n" + predicted_plain_text)
```

**Output:-**

```
SUNTZU=0.0010215425227561555
WHENYOUENGAGEINACTUALFIGHTINGIFVICTORYISLONGINCOMINGTHEMENSWEAPONSWILLGROWDULLANDTHEIRARDORWILLBEDAMPENEDIFYOULAYSIEGETOATOWNYOUWILLEXHAUST
THESTATEWILLNOTBEEQUALTOTHESTRAINNEVERFORGETWHENYOURWEAPONSAREDULLEDYOURARDORDAMPENEDYOURSTRENGTHEXHAUSTEDANDYOURTREASURESPENTOTHERCHIEFTAI
HOWEVERWISEWILLBEABLETOAVERTTHECONSEQUENCESTHATMUSTENSUETHUSTHOUGHWEHAVEHEARDOFSTUPIDHASTEINWARCLEVERNESSHASNEVERBEENSEENASSOCIATEDWITHLONG
FROMPROLONGEDWARFAREONLYONEWHOKNOWSTHEDISASTROUSEFFECTSOFALONGWARCANREALIZETHESUPREMEIMPORTANCEOFRAPIDITYINBRINGINGITTOACLOSEITISONLYONEWHO
STANDTHEPROFITABLEWAYOFCARRYINGITONTHESKILLFULGENERALDOESNOTRAISEASECONDLEVYNEITHERAREHISSUPPLYWAGONSLOADEDMORETHANTWICEONCEWARISDECLAREDHE
TURNHISARMYBACKFORFRESHSUPPLIESBUTCROSSESTHEENEMYSFRONTIERWITHOUTDELAYTHEVALUEOFTIMETHATISBEINGALITTLEAHEADOFYOUROPPONENTHASCOUNTEDFORMORET
TOCOMMISSARIAT
```

## Problem 2:-

```
frequency_in_english ={ "A": .08167, "B": .01492, "C": .02782, "D": .04253, "E": .12702, "F": .02228,
                        "G": .02015, "H": .06094, "I": .06996, "J": .00153, "K": .00772, "L": .04025,
                        "M": .02406, "N": .06749, "O": .07507, "P": .01929, "Q": .00095, "R": .05987,
                        "S": .06327, "T": .09056, "U": .02758, "V": .00978, "W": .02360, "X": .00150,
                        "Y": .01974, "Z": .00074 }

plain_text='ethicslawanduniversitypolicieswarningtodefendasystemyouneedtobeabletot\
hinklikeanattackerandthatincludesunderstandingtechniquesthatcanbeusedt\
ocompromisesecurityhoweverusingthosetechniquesintherealworldmayviolate\
thelawortheuniversitysrulesanditmaybeunethicalundersomecircumstancesev\
enprobingforweaknessesmayresultinseverepenaltiesuptoandincludingexpuls\
ioncivilfinesandjailtimeourpolicyineecsisthatyoumustrespecttheprivacya\
ndpropertyrightsofothersatalltimesorelseyouwillfailthecourseactinglawf\
ullyandethicallyisyourresponsibilitycarefullyreadthecomputerfraudandab\
useactcfaaafederalstatutethatbroadlycriminalizescomputerintrusionthisi\
soneofseverallawsthatgovernhackingunderstandwhatthelawprohibitsifindou\
btwecanreferyoutoanattorneypleasereviewitsspoliciesonresponsibleuseoft\
echnologyresourcesandcaenspolicydocumentsforguidelinesconcerningproper'


plain_text = plain_text.upper()
total_char = 26
```

```
# Below code is simple implementation of population variance using given english letter frequencies
sum_ = 0
for char in frequency_in_english:
  sum_ += frequency_in_english[char]
mean = sum_/total_char
variance = 0
for char in frequency_in_english:
  xi = frequency_in_english[char]
  variance += (xi-mean)**2
part_a = variance/total_char
print("a. Population variance of the relative letter frequencies in English text = "+str(part_a))



# this method will give us population variance of a given text.
def population_variance(cipher):
    frequency = [0]*total_char
    for i in range(0,len(cipher)):
        frequency[ord(cipher[i])-ord('A')] += 1
    sum_ = 0
    mean = 1/total_char
    for i in range(0,total_char):
        sum_ += (frequency[i]/len(cipher) - mean)**2
    return sum_/total_char

print("\nb. Population variance of the relative letter frequencies in given Plain text = "+str(population_variance(plain_text)))


# below code generate cipher text with given keys and calculate population variance for each cipher text
print('\nc. Population variance of the cipher text : ')
for key in ("YZ","XYZ","WXYZ","VWXYZ","UVWXYZ"):
    cipher_text = ""
    for i in range(0,len(plain_text)):
        cipher_text = cipher_text + chr((ord(plain_text[i])+ord(key[i%len(key)])-2*ord('A'))%26+ord('A'))
    print("(using "+ key + ") = " +str(population_variance(cipher_text)))


# this will calculate mean population variance of cipher text using all the keys
print('\nd. Mean population variance of the cipher text :')
for key in ("YZ","XYZ","WXYZ","VWXYZ","UVWXYZ"):
    cipher_text = ""
    # make cipher text
    for i in range(0,len(plain_text)):
        cipher_text = cipher_text + chr((ord(plain_text[i])+ord(key[i%len(key)])-2*ord('A'))%26+ord('A'))
    key_length = len(key)
    ciphers = []
    # break the cipher text into key length chunks
    for _ in range(0, key_length):
      ciphers.append("")
    for c in range(0,len(cipher_text)):
      ciphers[c%key_length] += cipher_text[c]
    # calculate variance for every chunk and mean the variances
    variance_sum = 0
    for i in range(0, len(ciphers)):
      variance_sum += population_variance(ciphers[i])
    print("(using "+ key + ") = " +str(variance_sum/len(ciphers)))




# Below code calculate mean population variance of the cipher text using UVWXYZ with different key length(2-5)
print("\ne. Mean population variance of the cipher text using UVWXYZ with different key length: ")
cipher_text = ""
key = "UVWXYZ"
for i in range(0,len(plain_text)):
    cipher_text = cipher_text + chr((ord(plain_text[i])+ord(key[i%len(key)])-2*ord('A'))%26+ord('A'))
for key_length in range(2, 6):
  ciphers = []
  # break the cipher text into key length chunks
  for _ in range(0, key_length):
    ciphers.append("")
  for c in range(0, len(cipher_text)):
    ciphers[c%key_length] += cipher_text[c]
  variance_sum = 0
   # calculate variance for every chunk and mean the variances
  for i in range(0, len(ciphers)):
    variance_sum += population_variance(ciphers[i])
  print("(using key length " + str(key_length) + " ) = " + str(variance_sum/key_length))
```

**Output:-**

```
a. Population variance of the relative letter frequencies in English text = 0.0010405667735207099

b. Population variance of the relative letter frequencies in given Plain text = 0.001006973795435334

c. Population variance of the ciphertext :
(using YZ) = 0.0005425572595902266
(using XYZ) = 0.0003474151001623527
(using WXYZ) = 0.0002459193736666264
(using VWXYZ) = 0.00018148975566557982
(using UVWXYZ) = 0.00016797152786163775

Explanation:- More non-uniform shifts in the plaintext are introduced with a longer key. This lowers variance by increasing the randomness o
So, the key with a small length has high variance but low randomness.


d. Mean population variance of the cipher text :
(using YZ) = 0.0011098867554911514
(using XYZ) = 0.0011034546954876623
(using WXYZ) = 0.0012505199318386133
(using VWXYZ) = 0.0011889247809577482
(using UVWXYZ) = 0.0013157126756027854

Explanation:- When I compare the variance to the other three parts, I find that part a, which was the population variance of relative lette
 This is due to the fact that the variance was computed across the set of characters in the key that were all moved by the same character. A


e. Mean population variance of the ciphertext using UVWXYZ with different key lengths:
(using key length 2 ) = 0.00046995129412711825
(using key length 3 ) = 0.0005082166002495672
(using key length 4 ) = 0.0005558574514618472
(using key length 5 ) = 0.0003314984771028727

Explanation:- This is a very effective Kasiski assault variation. We must break ciphertexts into blocks of a specific length and search for
population variance. We are confident that if we predict the key length incorrectly, the average of population variations will be significa
but if we guess the key length correctly, the average of population variances will almost certainly increase to 0.001 level.
```

**Explanation:-**

**c.** More non-uniform shifts in the plaintext are introduced with a longer key. This lowers variance by increasing the randomness of relative letter frequency. So, the key with a small length has high variance but low randomness.

**d.** When I compare the variance to the other three parts, I find that part a, which was the population variance of relative letter frequencies in English text, is comparable. This is due to the fact that the variance was computed across the set of characters in the key that were all moved by the same character. As a result, each set performs as a Caesar cipher.

e. This is a very effective Kasiski assault variation. We must break ciphertexts into blocks of a specific length and search for commonly occurring patterns in the Kasiski attack using population variance. We are confident that if we predict the key length incorrectly, the average of population variations will be significantly lower than the level of regular English texts, but if we guess the key length correctly, the average of population variances will almost certainly increase to 0.001 level.

## Problem 3:-

No, all the 56 bits of the DES key is used an unequal number of times in $K_i$. Like zeroth location, the bit is missing only once, in key 3 but other keys are missing more than once.

Here are the missing bits of the $K_i$ representing the key of round $i$

I'm assuming a bit position starting from 0 in 56 bits (0-55 locations).

| Round Key | Missing bits position from initial 56 bits in the $K_i$ |
|---|---|
| $K_1$ | 9 18 22 25 35 38 43 54 |
| $K_2$ | 10 19 23 26 36 39 44 55 |

| | |
|---|---|
| $K_3$ | 0 12 21 25 29 38 41 46 |
| $K_4$ | 2 14 23 27 31 40 43 48 |
| $K_5$ | 1 4 16 25 33 42 45 50 |
| $K_6$ | 3 6 18 27 35 44 47 52 |
| $K_7$ | 1 5 8 20 37 46 49 54 |
| $K_8$ | 3 7 10 22 28 39 48 51 |
| $K_9$ | 4 8 11 23 29 40 49 52 |
| K10 | 6 10 13 25 31 42 51 54 |
| K11 | 8 12 15 27 28 33 44 53 |
| K12 | 1 10 14 17 30 35 46 55 |
| K13 | 3 12 16 19 29 32 37 48 |
| K14 | 5 14 18 21 31 34 39 50 |
| K15 | 7 16 20 23 33 36 41 52 |
| K16 | 8 17 21 24 34 37 42 53 |