CS 411-507 Cryptography

Homework 1

Yiğit Kaan Tonkaz

29154

1-

def decrypt\_fun(ciphertext):

results = []

for i in range(26):

decrypted\_text = ''

for c in ciphertext:

decrypted\_character = chr(((ord(c) - ord('A') - i) % 26) + ord('A'))

decrypted\_text += decrypted\_character

results.append((i, decrypted\_text))

return results

results = decrypt\_fun("NLPDLC")

for i, decrypted\_text in results:

print(f"Shift {i}: {decrypted\_text}")

I defined a function called “decrypt\_fun” to decrypt the cipher text.

The output of code gives us the shifted texts in each shift and in shift 11 it gives us the meaningful word which is CAESAR.

Therefore, the encryption key is y = (x - 11) mod 26, and the decrypted word is "CAESAR."

2-

import math

import random

import fractions

ciphertext = "J gjg mxa czjq ayr arpa. J ulpa cxlmg ayerr ylmgerg rqrwrm hzdp ax gx ja hexmn."

ciphertext = ''.join(filter(str.isalpha, ciphertext)).lower()

counts = {}

for i in ciphertext:

if i.isalpha():

if i in counts:

counts[i] += 1

else:

counts[i] = 1

for i in sorted(counts):

count = counts[i]

print(f"{i}: {count}")

lowercase = {'a':0, 'b':1, 'c':2, 'd':3, 'e':4, 'f':5, 'g':6, 'h':7, 'i':8,

'j':9, 'k':10, 'l':11, 'm':12, 'n':13, 'o':14, 'p':15, 'q':16,

'r':17, 's':18, 't':19, 'u':20, 'v':21, 'w':22, 'x':23, 'y':24,

'z':25}

uppercase ={'A':0, 'B':1, 'C':2, 'D':3, 'E':4, 'F':5, 'G':6, 'H':7, 'I':8,

'J':9, 'K':10, 'L':11, 'M':12, 'N':13, 'O':14, 'P':15, 'Q':16,

'R':17, 'S':18, 'T':19, 'U':20, 'V':21, 'W':22, 'X':23, 'Y':24,

'Z':25}

inv\_lowercase = {0:'a', 1:'b', 2:'c', 3:'d', 4:'e', 5:'f', 6:'g', 7:'h', 8:'i',

9:'j', 10:'k', 11:'l', 12:'m', 13:'n', 14:'o', 15:'p', 16:'q',

17:'r', 18:'s', 19:'t', 20:'u', 21:'v', 22:'w', 23:'x', 24:'y',

25:'z'}

inv\_uppercase = {0:'A', 1:'B', 2:'C', 3:'D', 4:'E', 5:'F', 6:'G', 7:'H',

8:'I', 9:'J', 10:'K', 11:'L', 12:'M', 13:'N', 14:'O', 15:'P',

16:'Q', 17:'R', 18:'S', 19:'T', 20:'U', 21:'V', 22:'W', 23:'X',

24:'Y', 25:'Z'}

letter\_count = {'A':0, 'B':0, 'C':0, 'D':0, 'E':0, 'F':0, 'G':0, 'H':0, 'I':0,

'J':0, 'K':0, 'L':0, 'M':0, 'N':0, 'O':0, 'P':0, 'Q':0,

'R':0, 'S':0, 'T':0, 'U':0, 'V':0, 'W':0, 'X':0, 'Y':0, 'Z':0}

def egcd(a, b):

x,y, u,v = 0,1, 1,0

while a != 0:

q, r = b//a, b%a

m, n = x-u\*q, y-v\*q

b,a, x,y, u,v = a,r, u,v, m,n

gcd = b

return gcd, x, y

def modinv(a, m):

if a < 0:

a = m+a

gcd, x, y = egcd(a, m)

if gcd != 1:

return None

else:

return x % m

def Affine\_Dec(ptext, key):

plen = len(ptext)

ctext = ''

for i in range (0,plen):

letter = ptext[i]

if letter in lowercase:

poz = lowercase[letter]

poz = (key.gamma\*poz+key.theta)%26

#print poz

ctext += inv\_lowercase[poz]

elif letter in uppercase:

poz = uppercase[letter]

poz = (key.gamma\*poz+key.theta)%26

ctext += inv\_uppercase[poz]

else:

ctext += ptext[i]

return ctext

class key(object):

alpha=0

beta=0

gamma=0

theta=0

key.alpha = 11

key.beta = 25

key.gamma = modinv(key.alpha, 26) # you can compute decryption key from encryption key

key.theta = 26-(key.gamma\*key.beta)%26

print(Affine\_Dec(ciphertext, key))

First of all, both of sentences begin with J which might be ‘A’ or ‘I’ so it can be

0->9 or 8->9

E(0) = b mod 26 ≡ 9

b = 9 mod 26

or

E(8) = (8a+b) mod 26 ≡ 9

Secondly, most frequent letters are ‘a’ and ‘r’.

Therefore, ‘a’ or ‘r’ is encrypted of ‘t’.

Then it can be 19->0 or 19->17

First try a,

19 -> 0

So,

E(x) = (ax + b) mod 26

E(19) = (19a + b) mod 26 ≡ 0

19a + b = 0 mod 26

Try r,

19->17

So, E(19) = 19a + b mod 26 ≡ 17

Lets substitute these two

19a + b = 0 mod 26

b = 9 mod 26

* 19a = -9 mod 26 = 17 mod 26
* a = 5

95 + b = 0 mod 26

b = 9

it does not give meaningful sentence.

try these two

8a + b mod 26 ≡ 9

19a + b mod 26 ≡ 17

* 11a = (17-9) mod 26
* 11a = 8 mod 26
* a = 18
* 144 + b mod 26 ≡ 9
* 14 + b mod 26 ≡ 9
* b = 21

also it does not give a meaningful sentence

lets try

8a + b mod 26 = 9

19a + b mod 26 = 0

* 11a mod 26 = -9
* 11a mod 26 = 17
* 11a = 17 mod 26
* a = 11

88 + b mod 26 = 9

10 + b mod 26 = 9

1 + b mod 26 = 0

b = 25

E(x) = 11a + 25 mod 26 -> gives us a meaningful sentence

Therefore, keys are a = 11 and b = 25,

Plaintext is -> I did not fail the test. I just found three hundred eleven ways to do it wrong.

3-

First of all, let say k is length of plaintext,

like in the example,

19 x 30 + 7 = 577

We can say that

(a x 577 + b) mod m = c (according to the next biagram)

There are 30 possible characters therefore each letter in biagram has mod 30

So the mod of biagram should be 30 \* 30 = 900 -> mod 900

Key can be any number between 0 and 900 because 0 cannot change the plaintext

Let say key = 600

Then 577 + 600 mod 900 ≡ 1177 mod 900 ≡ 277 mod 900

Next, we need to convert 277 into the equivalent biagram

277/30 = 9.23 = 9 + 10/30

->9

->10

Which are JK

Therefore, the encrypted version of TH is JK

And the key space should be alphabet size ^ length -> 30 ^k

For this example, it equals 30^2 = 900

4-

Actually the affine cipher defined in question (3) is not secure enough against the letter frequency analysis because the key is constant. If the frequency distribution of letters is analyzed and figured out, it becomes vulnerable to frequency analysis attacks. In other words, it is so simple to understand due to letter substitution process. Also, It requires a very long key which is expensive to produce.

5-

6-

Due to one-time-pad, each key is going to used randomly and used for only once. And in one time pad, shift choice does not depend on the distribution of letters in the plaintext. Therefore, pβ is independent of the values of pα. Also, due to the uniform random shift, each letter in ciphertext has an equal chance of being chosen as the shift for any character in the plaintext. Therefore, pβ remains constant at 1/29 probability.

7-

First I try to find the key length with these code

import re

from collections import Counter

def find\_repeated\_sequences(ciphertext, sequence\_length=3):

repeated\_sequences = []

pattern = re.compile(r'(?=(\w{%d}))' % sequence\_length)

matches = re.finditer(pattern, ciphertext)

positions = {}

for match in matches:

sequence = match.group(1)

if sequence not in positions:

positions[sequence] = [match.start()]

else:

positions[sequence].append(match.start())

for sequence, pos\_list in positions.items():

if len(pos\_list) > 1:

intervals = [pos\_list[i+1] - pos\_list[i] for i in range(len(pos\_list) - 1)]

repeated\_sequences.append((sequence, intervals))

return repeated\_sequences

def gcd(a, b):

while b:

a, b = b, a % b

return a

def find\_key\_length(ciphertext):

repeated\_sequences = find\_repeated\_sequences(ciphertext, sequence\_length=3)

interval\_gcds = []

for \_, intervals in repeated\_sequences:

for i in range(len(intervals) - 1):

interval\_gcds.append(gcd(intervals[i], intervals[i+1]))

key\_length\_candidates = [k for k, v in Counter(interval\_gcds).items() if v >= 3]

return key\_length\_candidates

ciphertext = "FNZFFZZMLQQZVOGAXXH PZ UPU QXGIHU UY NWJXR AHBDLPOMK YOUPZM, VOZAYCD. J TGQH B XUIJJZM ARS XOAH, BZJ D JP AT GLWUTB LO EVDWF AL GRHUI. OKPGMC L NME IRU NKGLFHK DQ UTK JUEQX JI UTK PQJHKMVF, KKO L MABZ WIQ YOLDWE GLUFRZ OFMBZV BE ZCHZ AVZQ JZ YKUJZM. D OPHK OKF NRPH TWE, D OPHK NRNQ VZRQXK, RKPY UIH MABZV ZAA FQPI YJPFFOHHT IOOKPGZ FQPIOIJ XTE. D OPHK NRNQ MMHBF JZHEE JJQF NE HHO, FNJXHT O’QH MATB FFMYZG QQXCDQE ZJ KBHK ADJFN DQ UTKH, BFF LMRN ARY KBNOO ROQ’Y CHBDZ KUJLKN WIQS. CHSQ ZCHZ TGQH CDUPJIF ZCH TAAK IPD EJX, FMZ DW, JF CDOM PU TRV SUJG. JF’Y ALSEZ-MDUQ YJXQ, FNZB LZUR KPI ZJ PBWK DW IQXZ. L XMTO WP FXVYFX OI HVDUKH, BXEJVIM, O NKBXR NHU ALA ISAS CHSQ. GIG ZQZ D NOAC OKBF O VP PZRT JPUTB WP M MMDWQEVUE, NAO LU’E G HRTF VMHDUUPV HDGQHZMXY, WIMZ’N ZIMZ DW JE. VMHDUUPV BDK OKF PKVG UTGO OJQ ZCHSQ, KQHSK YOROQ UQHS FNZP TBKVNT AL NXDT HPUOUTB OJRK DQ UTK KDTF, UA VVON KDTEOJQBFK ADJFN DQ UTKDU XAXF, WIQOM WSGZC, WIQOM VUDABJMQ GIG UTKDU TOOZQDQ, ZCDU U QIRX U YCDMX LVOM AT OKF SXJXOP GIG LUYN WIAYZ VUATZV BZJ RHFB UQHS FNZP; UTUPJI U’S XROHOIFFP OI PZ TKVUU FNVW JF’Y GROS HZHO ZUOKJZM WXU M MMDWQEVUE. MTY L TTGGO OAZ RHFB LMRN PKNSBUX, WXU EOHSMK HZFBGYZ L TTGGO CQ NVSQK OI PZ FKVUT, U YCDMX YOHFB ST VPGR DQ FYUOLPZ. O GRWQ ZCH TFOXNZ XKVYFE OI VQDOIJ, UTK WOVQ YFB - UTGO’V BXR DW JE. OO’V OAZ V PBFZZU PR OIWFXRZFU AX GRHUI, DW’T XUQLOS CDWI ATZ’V JZYDGF, IOOK PZK’N VUASVFI."

ciphertext = ''.join(filter(str.isalpha, ciphertext)).upper()

key\_length\_candidates = find\_key\_length(ciphertext)

print("Possible key lengths:", key\_length\_candidates)

It gives me the output of

Possible key lengths: [5, 15, 30]

Then with the code below, we can generate random keys

import random

import string

def generate\_vigenere\_key(key\_length):

key = ''.join(random.choice(string.ascii\_lowercase) for \_ in range(key\_length))

return key

key\_length = 5

key = generate\_vigenere\_key(key\_length)

print("Generated Vigenère Key:", key)

After a long process, I found the key “mgvdb” and if we put it on this code

def vigenere\_decrypt(ciphertext, key):

decrypted\_text = ""

key\_length = len(key)

for i in range(len(ciphertext)):

if ciphertext[i].isalpha():

key\_char = key[i % key\_length].lower()

shift = ord(key\_char) - ord('a')

if ciphertext[i].isupper():

decrypted\_char = chr(((ord(ciphertext[i]) - ord('A') - shift) % 26) + ord('A'))

else:

decrypted\_char = chr(((ord(ciphertext[i]) - ord('a') - shift) % 26) + ord('a'))

decrypted\_text += decrypted\_char

else:

decrypted\_text += ciphertext[i]

return decrypted\_text

ciphertext = "FNZFFZZMLQQZVOGAXXH PZ UPU QXGIHU UY NWJXR AHBDLPOMK YOUPZM, VOZAYCD. J TGQH B XUIJJZM ARS XOAH, BZJ D JP AT GLWUTB LO EVDWF AL GRHUI. OKPGMC L NME IRU NKGLFHK DQ UTK JUEQX JI UTK PQJHKMVF, KKO L MABZ WIQ YOLDWE GLUFRZ OFMBZV BE ZCHZ AVZQ JZ YKUJZM. D OPHK OKF NRPH TWE, D OPHK NRNQ VZRQXK, RKPY UIH MABZV ZAA FQPI YJPFFOHHT IOOKPGZ FQPIOIJ XTE. D OPHK NRNQ MMHBF JZHEE JJQF NE HHO, FNJXHT O’QH MATB FFMYZG QQXCDQE ZJ KBHK ADJFN DQ UTKH, BFF LMRN ARY KBNOO ROQ’Y CHBDZ KUJLKN WIQS. CHSQ ZCHZ TGQH CDUPJIF ZCH TAAK IPD EJX, FMZ DW, JF CDOM PU TRV SUJG. JF’Y ALSEZ-MDUQ YJXQ, FNZB LZUR KPI ZJ PBWK DW IQXZ. L XMTO WP FXVYFX OI HVDUKH, BXEJVIM, O NKBXR NHU ALA ISAS CHSQ. GIG ZQZ D NOAC OKBF O VP PZRT JPUTB WP M MMDWQEVUE, NAO LU’E G HRTF VMHDUUPV HDGQHZMXY, WIMZ’N ZIMZ DW JE. VMHDUUPV BDK OKF PKVG UTGO OJQ ZCHSQ, KQHSK YOROQ UQHS FNZP TBKVNT AL NXDT HPUOUTB OJRK DQ UTK KDTF, UA VVON KDTEOJQBFK ADJFN DQ UTKDU XAXF, WIQOM WSGZC, WIQOM VUDABJMQ GIG UTKDU TOOZQDQ, ZCDU U QIRX U YCDMX LVOM AT OKF SXJXOP GIG LUYN WIAYZ VUATZV BZJ RHFB UQHS FNZP; UTUPJI U’S XROHOIFFP OI PZ TKVUU FNVW JF’Y GROS HZHO ZUOKJZM WXU M MMDWQEVUE. MTY L TTGGO OAZ RHFB LMRN PKNSBUX, WXU EOHSMK HZFBGYZ L TTGGO CQ NVSQK OI PZ FKVUT, U YCDMX YOHFB ST VPGR DQ FYUOLPZ. O GRWQ ZCH TFOXNZ XKVYFE OI VQDOIJ, UTK WOVQ YFB - UTGO’V BXR DW JE. OO’V OAZ V PBFZZU PR OIWFXRZFU AX GRHUI, DW’T XUQLOS CDWI ATZ’V JZYDGF, IOOK PZK’N VUASVFI."

ciphertext = ''.join(filter(str.isalpha, ciphertext)).upper()

key = "mgvdb"

decrypted\_text = vigenere\_decrypt(ciphertext, key)

print(decrypted\_text)

We get the output of this result

THECENTRIPETALFORCEONOURPLANETISSTILLFEARFULLYSTRONGALYOSHAIHAVEALONGINGFORLIFEANDIGOONLIVINGINSPITEOFLOGICTHOUGHIMAYNOTBELIEVEINTHEORDEROFTHEUNIVERSEYETILOVETHESTICKYLITTLELEAVESASTHEYOPENINSPRINGILOVETHEBLUESKYILOVESOMEPEOPLEWHOMONELOVESYOUKNOWSOMETIMESWITHOUTKNOWINGWHYILOVESOMEGREATDEEDSDONEBYMENTHOUGHIVELONGCEASEDPERHAPSTOHAVEFAITHINTHEMYETFROMOLDHABITONESHEARTPRIZESTHEMHERETHEYHAVEBROUGHTTHESOUPFORYOUEATITITWILLDOYOUGOODITSFIRSTRATESOUPTHEYKNOWHOWTOMAKEITHEREIWANTTOTRAVELINEUROPEALYOSHAISHALLSETOFFFROMHEREANDYETIKNOWTHATIAMONLYGOINGTOAGRAVEYARDBUTITSAMOSTPRECIOUSGRAVEYARDTHATSWHATITISPRECIOUSARETHEDEADTHATLIETHEREEVERYSTONEOVERTHEMSPEAKSOFSUCHBURNINGLIFEINTHEPASTOFSUCHPASSIONATEFAITHINTHEIRWORKTHEIRTRUTHTHEIRSTRUGGLEANDTHEIRSCIENCETHATIKNOWISHALLFALLONTHEGROUNDANDKISSTHOSESTONESANDWEEPOVERTHEMTHOUGHIMCONVINCEDINMYHEARTTHATITSLONGBEENNOTHINGBUTAGRAVEYARDANDISHALLNOTWEEPFROMDESPAIRBUTSIMPLYBECAUSEISHALLBEHAPPYINMYTEARSISHALLSTEEPMYSOULINEMOTIONILOVETHESTICKYLEAVESINSPRINGTHEBLUESKYTHATSALLITISITSNOTAMATTEROFINTELLECTORLOGICITSLOVINGWITHONESINSIDEWITHONESSTOMACH

Which seems meaningful

Therefore, key is mgvdb and key length is 5