Math3302 Assignment 2

Dominic Scocchera, s4642675, Tute 1

May 2023

Q1

a)

We want to calculate the number of keys in an affine cipher which has a plaintext alphabet of size 36.

$$\mathcal{K} = \{(a, b) \in \mathbb{Z}_{36} \times \mathbb{Z}_{36} : \gcd(a, 36)\}$$

From this we know that $a \in \{1, 3, 5, 7, 9, 11, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35\}$. For each of these 17 values that a can take there are 36 possible values of b that a given value of a can pair with, hence:

$$|\mathcal{K}| = 17 \cdot 36 = 612$$

b)

We know that $E_k(x) = ax + b \mod 36$. From the given plaintext and cipher text also know that:

$$\begin{aligned} 25 &= 5a + b \mod 36 \\ 29 &= 25a + b \mod 36 \\ 32 &= 22a + b \mod 36 \end{aligned}$$

Furthermore we get:

$$\begin{aligned} 25 - 5a &= b \mod 36 \\ 29 - 25a &= b \mod 36 \end{aligned}$$

Hence:

$$\begin{aligned} 25 - 5a &= 29 - 25a \mod 36 \\ 20a &= 4 \mod 36 \end{aligned}$$

So a=11 or 29, and b=6 or 24. Only the pair (11,6) satisfies $32 = 22a + b \mod 36$, hence it is our key. To decrypt we use the function $D_k(y) = 11^{-1}(y-6) \mod 36 = 23(y-6) \mod 36$. Applying this to the cipher text PTW8118UA, we get the plaintext is 5PMATTACK.

Q2

a)

$$\begin{aligned}
K^{-1} &= \det(K)^{-1} \begin{pmatrix} 4 & -3 \\ -3 & 6 \end{pmatrix} \mod 26 \\
&= (6 \cdot 4 - 3 \cdot 3)^{-1} \begin{pmatrix} 4 & -3 \\ -3 & 6 \end{pmatrix} \mod 26 \\
&= 15^{-1} \begin{pmatrix} 4 & -3 \\ -3 & 6 \end{pmatrix} \mod 26 \\
&= 7 \begin{pmatrix} 4 & -3 \\ -3 & 6 \end{pmatrix} \mod 26 \\
&= \begin{pmatrix} 2 & 5 \\ 5 & 16 \end{pmatrix}
\end{aligned}$$

b)

WFFUBT corresponds to the numbers $22, 5, 5, 20, 1, 19$.

$$\begin{aligned}
K^{-1} \begin{pmatrix} 22 \\ 5 \end{pmatrix} &= \begin{pmatrix} 17 \\ 8 \end{pmatrix} \\
K^{-1} \begin{pmatrix} 5 \\ 20 \end{pmatrix} &= \begin{pmatrix} 6 \\ 7 \end{pmatrix} \\
K^{-1} \begin{pmatrix} 1 \\ 19 \end{pmatrix} &= \begin{pmatrix} 19 \\ 23 \end{pmatrix}
\end{aligned}$$

So the numbers for the plain text are $17, 8, 6, 7, 19, 23$ and hence the plaintext is RIGHTX.

Q3

a)

First we implement the index of coincidence:

$$\varphi = \frac{\sum_{i=1}^{k} n_i(n_i - 1)}{n(n-1)}$$

Where n_i are the counts of the letters in the cipher text and n is the total length of the cipher text. And also Friedman's method 2 for working out the key length:

$$m \approx \frac{n(\varphi_L - \varphi_0)}{(n-1)\varphi(T) - n\varphi_0 + \varphi_L}$$

Where m is the length of the key, φ_L is the index of coincidence for english (0.0667), φ_0 is the index of coincidence for random text $\left(\frac{1}{26}\right)$ and $\varphi(T)$ is the index of coincidence for the entire cipher text.

```python
import collections
import numpy as np
import string

cipher = "The cipher text that was given (Too long to display)"
def phi(c):
    x = list(collections.Counter(c).items())
    n = len(c)
    phi = 0
    for i in range(len(x)):
        phi += x[i][1]*(x[i][1]-1)
    phi = phi/(n*(n-1))
    return phi

def friedmanMethod2(c):
    n = len(c)
    phi_0 = 1/26
    phi_L = 0.0667
    phi_T = phi(c)
    m = (n*(phi_L-phi_0))/((n-1)*phi_T-n*phi_0+phi_L)
    return m

print(friedmanMethod2(cipher))
```

From this we get $m \approx 2.8622543040047796$.

b)

For Friedman's method 1 we first split the cipher up into m arrays where each array contains the letters from the cipher that are in a position in the cipher congruent to $i \mod m, \quad i \in \{0, ..., m-1\}$. We then calculate the index of coincidence for each array, for multiple values of m (In the code we calculate it for $1 \leq m \leq 10$). Then we calculate the error via $\sum_{i=1}^{m} |\varphi_i - \varphi_L|$ where φ_i is the index of coincidence for a given m and i_{th} array. Repeating this over multiple values of n we then take the value of m that has the lowest error to be the most likely key length.

```python
def array2string(arr):
    strng = ""
    for i in range(len(arr)):
        strng += arr[i]
    return strng

def string2array(strng):
    array = np.array([])
    for i in range(len(strng)):
        array = np.append(array,strng[i])
    return array

def cipherSplit(m,c):
    split = []
    cipher = []
    for i in range(len(c)):
        cipher.append(c[i])
    cipher = np.array(cipher)
    add = m - (len(cipher)%m)
    for j in range(add):
        cipher = np.append(cipher,np.array([""]))
    cipher = cipher.reshape(int((len(cipher))/m),m)
    for k in range(m):
        split.append(array2string(cipher[:,k]))
    return split

def friedmanMethod1(c,maxm):
    phimat = np.zeros([maxm,maxm])
    loss = np.tril(0.0667*np.ones([maxm,maxm]))
    for i in range(1,maxm+1):
        splits = cipherSplit(i, c)
        for j in range(len(splits)):
            phimat[i-1,j] += phi(splits[j])
    error = np.abs(phimat-loss)
    error = np.sum(error,axis=1)
    m = np.where(error == np.min(error))[0][0]+1
    return m

print(friedmanMethod1(cipher,10))
```

From this we get that $m = 5$.

c)

Now we know the key length is probably 5 we can use frequency analysis to determine the key and then decrypt using $d_k((y_1, ..., y_m)) = (y_1 - k_1, ..., y_m - k_m)$. For the frequency analysis we assumed E is the most frequent and then took the first few most frequent letters in each of the m arrays and calculated the shift. This narrowed down the set of possible keys to something that could be guess and checked by hand to find something reasonable.

```
def letterFrequencies(x):
    freq = {}

    for i in x:
        if i in freq:
            freq[i] += 1
        else:
            freq[i] = 1
    total = sum(freq.values(), 0.0)
    freq = {k: v / total for k, v in freq.items()}
    return freq
```

```
def shiftCalculation(c,m):
    d = {chr(i+65):i for i in range(0,26)}
    letters = ["E","T","A","O","I","N","S","H","R","D","L","U","C",
                                      "M","W","F","G","Y","P","B","
                                      V","K","J","Z","X","Q"] #
                                      Letters from highest
                                      frequency in english language
                                       to lowest frequency
    split = cipherSplit(m, c)
    shifts = []
    for i in range(len(split)):
        listletters = []
        freq = letterFrequencies(split[i])
        sort = sorted(freq.values(),reverse=True)
        for j in range(len(freq)):
            listletters.append(d[list(freq.keys())[list(freq.values
                                              ()).index(sort[j])]])
        distance = []
        for k in range(4):
            ma = max(d[letters[0]],listletters[k])
            mi = min(d[letters[0]],listletters[k])
            distance.append((ma-mi))
        shifts.append(distance)
    return shifts

shifts = shiftCalculation(cipher,5)

#the key is [17,8,21,4,17]

def decrypt(c,m):
    d = {chr(i+65):i for i in range(0,26)}
    split = cipherSplit(m, c)
    shifts = [17,8,21,4,17]
    arraylen = max(len(i) for i in split)
    decrypt = []
    for i in range(len(split)):
        strng = ""
        for j in range(len(split[i])):
            s = (d[split[i][j]]-shifts[i])%26
            strng += list(d.keys())[list(d.values()).index(s)]
        decrypt.append(strng)
    message = string2array(decrypt[0])
    if len(message)!=arraylen:
        while len(message)!=arraylen:
            message = np.append(message,0)
    for k in range(len(decrypt)-1):
        arr = string2array(decrypt[k+1])
        if len(arr)!=arraylen:
            while len(arr)!=arraylen:
                arr = np.append(arr,0)
        message = np.vstack((message,string2array(decrypt[k+1])))
    message = message.T
    shape = message.shape[0]*message.shape[1]
    message = message.reshape(shape)
    return message
```

```
message = decrypt(cipher,5)
print(array2string(message))
```

From this we get that the two most likely shifts for the 5 arrays were $[[13, 17], [8, 3], [17, 21], [4, 0], [17, 21]]$. From guessing on this information we find that the key was RIVER and the plain text is MYFATHERSFAMILYNAMEBEINGPIRRIPANDMYCHRISTIANNAMEPHILIP-MYINFANTTONGUE COULDMAKEOFBOTHNAMESNOTHINGLONGEROR-MOREEXPLICITTHANPIPSOICALLEDMYSELF PIPANDCAMETOBECALLED-PIPIGIVEPIRRIPASMYFATHERSFAMILYNAMEONTHEAUTHORITY OFHIS-TOMBSTONEANDMYSISTERMRSJOEGARGERYWHOMARRIEDTHEBLACK-SMITHASINEVER SAWMYFATHERORMYMOTHERANDNEVERSAWANY-LIKENESSOFEITHEROFTHEMFORTHEIRDAYS WERELONGBEFORETHE-DAYSOFPHOTOGRAPHSMYFIRSTFANCIESREGARDINGWHATTHEYWERE LIKEWEREUNREASONABLYDERIVEDFROMTHEIRTOMBSTONESTHE-SHAPEOFTHELETTERSONMY FATHERSGAVEMEANODDIDEATHATHE-WASASQUARESTOUTDARKMANWITHCURLYBLACKHAIR FROMTHECHAR-ACTERANDTURNOFTHEINSCRIPTIONALSOGEORGIANAWIFEOFTHEABOVE IDREWACHILDISHCONCLUSIONTHATMYMOTHERWASFRECKLEDAND-SICKLYTOFIVELITTLE STONELOZENGESEACHABOUTAFOOTANDAHALF-LONGWHICHWEREARRANGEDINANEATROW BESIDETHEIRGRAVEANDW-ERESACREDTOTHEMEMORYOFFIVELITTLEBROTHERSOF MINEWHOGAVE-UPTRYINGTOGETALIVINGEXCEEDINGLYEARLYINTHATUNIVERSAL STRUG-GLEIAMINDEBTEDFORABELIEFIRELIGIOUSLYENTERTAINEDTHATTHEY-HADALLBEEN BORNONTHEIRBACKSWITHTHEIRHANDSINTHEIRTROUSER-SPOCKETSANDHADNEVERTAKENTHE MOUTINTHISSTATEOFEXISTENCEOUR-SWASTHEMARSHCOUNTRYDOWNBYTHERIVERWI THINASTHERIVER-WOUNDTWENTYMILESOFTHESEAMYFIRSTMOSTVIVIDANDBROAD IM-PRESSIONOFTHEIDENTITYOFTHINGSSEEMSTOMETOHAVEBEENGAINE-DONAMEMORABLERAW AFTERNOONTOWARDSEVENINGATSUCHA-TIMEIFOUNDOUTFORCERTAINTHATTHISBLEAKPLACEOVER GROWN-WITHNETTLESWASTHECHURCHYARDANDTHATPHILIPPIRRIPLATEOFTHISPARISHANDAL SOGEORGIANAWIFEOFTHEABOVEWEREDEADANDBURIEDANDTHATALEXAN-DERBARTHOLOMEWABRA HAMTOBIASANDROGERINFANTCHILDRENOFTHEAFORE-SAIDWEREALSODEADANDBURIEDAND THATTHEDARKFLATWILDER-NESSBEYONDTHECHURCHYARDINTERSECTEDWITHDYKESANDMOUND SANDGATESWITHSCATTEREDCATTLEFEEDINGONITWASTHEMARSH-ESANDTHATTHELOW LEADENLINEBEYONDWASTHERIVERANDTHATTHEDIS-TANTSAVAGELAIRFROMWHICHTHEWIND WASRUSHINGWASTHESEAANDTHATTHES-MALLBUNDLEOFSHIVERSGROWINGAFRAIDOFITALLANDBEGINNING TOCRYWASPIPHOLDYOURNOISECRIEDATERRIBLEVOICEASAMANSTART-EDUPFROMAMONGTHEGRAVE SATTHESIDEOFTHECHURCHPORCH-KEEPSTILLYOULITTLEDEVILORILLCUTYOURTHROATAFEARFUL MAN-ALLINCOARSEGREYWITHAGREATIRONONHISLEGAMANWITHNOHATAND-WITH

Q4

History

The ADFGX cipher was invented by Colonel Fritz Nebel, a communications officer of the Kaiser's army in world war one and went into usage on the 5th of March 1918. Georges Painvin broke the cipher in early June 1918.

Cipher

The first part of the cipher is controlled by the polybius square, whose columns and rows are labelled ADFGVX. These letters were chosen because they are hard to mess up in morse code, as this was the communication technology used in world war one. To fill in the table we go from left to right, top to bottom. We start by filling it out with a key word, leaving out any repeat letters in the key word. Then we fill it in with the remaining unused letters of the alphabet in alphabetical order with the digits 1 to 9 being placed after A (1), B(2), C(3),... and the digit 0 being placed after J. As an example this would be the polybius square for the key word MATHEMATICS:

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

The plaintext is then encoded into the coordinates of the given letter or number in the table, for example SECRETCODE becomes FA,DA,DV,VX,DA,AG,DV,VF,FG,DA. Then given a key, for example FIVEX, we arrange the text as below:

F	I	V	E	X
FA	DA	DV	VX	DA
AG	DV	VF	FG	DA

Now we rearrange the columns in alphabetical order:

E	F	I	V	X
VX	FA	DA	DV	DA
FG	AG	DV	VF	DA

Hence the cipher text becomes the columns in order, i.e. VX, FG, FA, AG, DA, DV, DV, VF, DA, DA. If the key for the table and key for permuting the text is known then decrypting it is the same as encryption but in reverse.

Information on the cipher obtained from Codes and Codebreakers In World War I, Greg Goebel, 3 May 2010,
https://web.archive.org/web/20100503103848/http://www.vectorsite.net/ttcode$_0 4.htmlm3$

Q5

a)

$$\begin{aligned}
a &= b^{-1} \mod \varphi(n) \\
&= (3609)^{-1} \mod \varphi(59 \cdot 71) \\
&= (3609)^{-1} \mod (59-1)(71-1) \\
&= (3609)^{-1} \mod 4060
\end{aligned}$$

From Eulers thereom ($a^{\varphi(n)} = 1 \mod n$) we know $(3609)^{-1} \mod 4060 = (3609)^{1344-1} \mod 4060 = 9$. Now $d_k(121) = (121)^9 \mod 4189 = 2514$. Now we show the working for the exponentiation:

$$\begin{aligned}
(3609)^{1344-1} \mod 4060 &= (3609)^{1024}(3609)^{256}(3609)^{32}(3609)^{16} \\
&\quad (3609)^{8}(3609)^{4}(3609)^{2}(3609)^{1} \mod 4060 \\
(3609)^{1} &= 3609 \mod 4060 \\
(3609)^{2} &= 401 \mod 4060 \\
(3609)^{4} &= (401)^2 \mod 4060 = 2461 \mod 4060 \\
(3609)^{8} &= (2461)^2 \mod 4060 = 3061 \mod 4060 \\
(3609)^{16} &= (3061)^2 \mod 4060 = 3301 \mod 4060 \\
(3609)^{32} &= (3301)^2 \mod 4060 = 3621 \mod 4060 \\
(3609)^{64} &= (3621)^2 \mod 4060 = 1901 \mod 4060 \\
(3609)^{128} &= (1901)^2 \mod 4060 = 401 \mod 4060 \\
(3609)^{256} &= (401)^2 \mod 4060 = 2461 \mod 4060 \\
(3609)^{512} &= (2461)^2 \mod 4060 = 3061 \mod 4060 \\
(3609)^{1024} &= (3061)^2 \mod 4060 = 3301 \mod 4060 \\
(3609)^{1343} \mod 4060 &= 3609 \cdot 401 \cdot 2461^2 \cdot 3061 \cdot 3301^2 \cdot 3621 \mod 4060 \\
&= 3609 \cdot 401 \cdot 3061 \cdot 3061 \cdot 3621 \cdot 3621 \mod 4060 \\
&= 3609 \cdot 401 \cdot 3061^2 \cdot 3621^2 \mod 4060 \\
&= 3609 \cdot 401 \cdot 3301 \cdot 1901 \mod 4060 \\
&= 9 \mod 4060
\end{aligned}$$

$$\begin{aligned}
(121)^9 \mod 4189 &= (121)^8(121)^1 \mod 4189\\
(121)^1 \mod 4189 &= 121 \mod 4189\\
(121)^2 \mod 4189 &= 2074 \mod 4189\\
(121)^4 \mod 4189 &= (2074)^2 \mod 4189 = 3562 \mod 4189\\
(121)^8 \mod 4189 &= (3562)^2 \mod 4189 = 3552 \mod 4189\\
(121)^9 \mod 4189 &= 3552 \cdot 121 \mod 4189\\
&= 2514 \mod 4189
\end{aligned}$$

b)

$b = \frac{\varphi(n)}{2} + 1$ is not a good choice as it is it's own inverse working mod $\varphi(n)$:

$$\begin{aligned}
b^2 \mod \varphi(n) &= \left(\frac{\varphi(n)}{2} + 1\right)^2 \mod \varphi(n)\\
&= \frac{\varphi(n)^2}{4} + \varphi(n) + 1 \mod \varphi(n)\\
&= 4^{-1} \mod \varphi(n) \cdot \varphi(n)^2 \mod \varphi(n) + \varphi(n) \mod \varphi(n) + 1 \mod \varphi(n)\\
&= 4^{-1} \mod \varphi(n) \cdot 0 + 1 \mod \varphi(n)\\
&= 1 \mod \varphi(n)
\end{aligned}$$

This makes it extremely easy to guess the key.

Q6

a)

The message in this cryptosystem is basically masked by multiplying it by $\beta^d \mod p = (\alpha^a)^d \mod p$, which means that 0 does not get masked at all as anything times 0 is 0. So it is a bad choice for a message word as no encryption actually occurs.

b)

Our opponent knows the public keys, the cipher texts and one of the messages, i.e. they know (p, α, β), $(\alpha^d, \beta^d x_1)$, $(\alpha^d, \beta^d x_2)$, x_1. To avoid finding d the opponent can calculate $x_1^{-1} \mod p$ via the approach for finding inverses used in Q5a), i.e. applying Euler's theorem and fast exponentiation. Then as $\beta^d x_1$ is known we multiply this value by x_1^{-1} to obtain β^d. We can again find the inverse of β^d mod p via the previously suggested method. Finally multiply $\beta^d x_2$ by $(\beta^d)^{-1}$ to obtain x_2.