Cryptography Coursework

Outline

We face 3 different ciphers, Vigenere for the 1st cipher, an XOR loop of a 2 letter key for the second one and for the last one a multiple step encryption, beginning with the same method used for the 2nd cipher and no conclusions on how the last part was encrypted.

Solution for Cipher 1

The competition is for papers that were published between December and October , and that show an outstanding contribution to cybersecurity science. The competition was created to stimulate research toward the development of systems that are resilient to cyber attacks. Entries are judged on scientific merit, the strength and significance of the work reported.

Cipher 1 Cryptanalysis

The look of the cipher can seems to be of either a shift encryption(Ceasar), and advanced ceasar or a vigenere. Let's start with these 3 encryption algorithms.

First step is to check for a shift encryption, for that we use the "basic" index of coincidence:

```
basic_IC = 0.04933009857277119
```

This means it isn't a simple shifted encryption (because otherwise we would be close to 0.66) and that the key length looks to be between 2 and 3 letters long [Arizone State University]. After that we can try to find the real key length using the advanced IC, this results in 3 possible results (I started with 10 letter long possible keys and if need I would have extended that length):

```
key length: 3
1.8586138613861387

key length: 6
1.8854421768707486

key length: 9
1.9368686868686866
```

For the advanced IC since we multiplied by 26 the basic IC we are looking for scores around 1.73.

For each possible length we divide the cipher into rows of the key length to obtain a matrix where each column has been encrypted with the same key and therefor we can attack each column as a ceasar cipher by using correlation attacks. These attacks consists in multiplying the letters by their possible shifted frequency values and make stand out the correct shifts. After printing all the correlation values, we have to keep the highest values. We do that for all columns and then we combine them together to find the proprer key.

```
1st column

4 : 0.050446300000000006
15 : 0.0636033

2nd column

4 : 0.04827560000000001
8 : 0.04926860000000001
19 : 0.07587880000000001

3rd column

6 : 0.07303079999999999
17 : 0.0503147
21 : 0.0528133
```

Here we are lucky after the first try with the best correlation indexes with find the key (15,19,6 = PTG)

Solution for Cipher 2

A free market does not require the existence of competition however it does require a framework that allows new market entrants. Hence in the lack of coercive barriers, and in markets with low entry cost it is generally understood that competition flourishes in a free-market environment. Typically, a modern free market economy would include other features, such as a stock exchange and a financial services sector, but they do not define it. Critics of the free market have argued that, in real world situations, it has proven to be susceptible to the development of price fixing monopolies. Such reasoning has led in the past to government intervention. It is fair to say that only one known example of a true free market exists and that is the Black Market. That is to say, anyone can produce anything at any time, and anyone can purchase anything available at any time.

Cipher 2 Cryptanalysis

I first thought that a good way to resolve ascii ciphers might have been a vigenere encryption using the ascii alphabets instead of regular ones but then after a lot of coding realized this wasn't possible.

After looking if it could be a simple substitution, the problem is that there are too many different characters. Therefore none of the tools used for cipher 1 are useful here.

We have the hint that the 1st letter of the plaintext is A. After understanding that for this type of cipher the most probable encryption is a repeated XOR encryption loop with the same key. And since we have the 1st letter of the plaintext we can obtain the 1 letter of the key easily.

I used online tools to help me figure out the possible lengths of the key: the highest possibility was for a key of 2 then 4, 6, 8 ... (https://wiremask.eu/tools/xor-cracker/) After doing some informative guessing and trying a couple of possible letters after an "A" at the beginning of a sentence I finally find the key: "7Z". The decryption is pretty straight forward.

Solution for Cipher 3

Only partial solution:

['SIKGMBLSSILCKVLEJCPSJNIUTOKJPAAAQCPAGGQHATGVPSEKMBKRAUVRJRTOETJOKOPQLEIRLQKRWNVRROWHQHKCKO',
'SMKCMFLWSMLGKRLAJGPWJJIQTKKNPEAEQGPEGCQLAPGRPWEOMFKVAQVVJVTKEPJKKKPULAIVLUKVWJVVRKWLQLKGKK',
'SOKAMDLUSOLEKPLCJEPUJHISTIKLPGAGQEPGGAQNARGPPUEMMDKTASVTJTTIERJIKIPWLCITLWKTWHVTRIWNQNKEKI',
'PIHGNBOSPIOCHVOEICSSINJUWOHJSABARCSADGRHBTDVSSFKNBHRBUURIRWOFTIOHOSQOEJROQHRTNURQOTHRHHCHO',
'PMHCNFOWPMOGHROAIGSWIJJQWKHNSEBERGSEDCRLBPDRSWFONFHVBQUVIVWKFPIKHKSUOAJVOUHVTJUVQKTLRLHGHK',
'POHANDOUPOOEHPOCIESUIHJSWIHLSGBGRESGDARNBRDPSUFMNDHTBSUTITWIFRIIHISWOCJTOWHTTHUTQITNRNHEHI',
'QIIGOBNSQINCIVNEHCRSHNKUVOIJRACASCRAEGSHCTEVRSGKOBIRCUTRHRVOGTHOIORQNEKRNQIRUNTRPOUHSHICIO',
'QMICOFNWQMNGIRNAHGRWHJKQVKINRECESGREECSLCPERRWGOOFIVCQTVHVVKGPHKIKRUNAKVNUIVUJTVPKULSLIGIK',
'QOIAODNUQONEIPNCHERUHHKSVIILRGCGSERGEASNCREPRUGMODITCSTTHTVIGRHIIIRWNCKTNWITUHTTPIUNSNIEII',
'WIOGIBHSWIHCOVHENCTSNNMUPOOJTAEAUCTACGUHETCVTSAKIBOREURRNRPOATNOOOTQHEMRHQORSNRRVOSHUHOCOO',
'WMOCIFHWWMHGORHANGTWNJMQPKONTEEEUGTECCULEPCRTWAOIFOVEQRVNVPKAPNKOKTUHAMVHUOVSJRVVKSLULOGOK',
'WOOAIDHUWOHEOPHCNETUNHMSPIOLTGEGUETGCAUNERCPTUAMIDOTESRTNTPIARNIOITWHCMTHWOTSHRTVISNUNOEOI',
'UIMGKBJSUIJCMVJELCVSLNOUROMJVAGAWCVAAGWHGTAVVSCKKBMRGUPRLRROCTLOMOVQJEORJQMRQNPRTOQHWHMCMO',
'UMMCKFJWUMJGMRJALGVWLJOQRKMNVEGEWGVEACWLGPARVWCOKFMVGQPVLVRKCPLKMKVUJAOVJUMVQJPVTKQLWLMGMK',
'UOMAKDJUUOJEMPJCLEVULHOSRIMLVGGGWEVGAAWNGRAPVUCMKDMTGSPTLTRICRLIMIVWJCOTJWMTQHPTTIQNWNMEMI']

Cipher 3 Cryptanalysis

After the second one we are pretty confident that this one uses a looped XOR encryption as well. I created a bruteforce to test out different key lengths by trying 90(cipher lenght) * 128 ^ key_length possibilities and keeping the alphabetical results. After finding that the best possibility is a 2 letter long key and trying all the possible decryptions methods for alphabetical cihpers, I haven't been successful in decryptiing the last cihper.

Appendix A

References:

```
[Arizone State University]
https://docs.google.com/presentation/d/12nESJSKTn0oAFJ0C17oIMZcywgnpL7cCeLxsmn6fZro
/pub?start=false&pageId=111414058055695777099#slide=id.g3521cc1e5_118
```

Global variables

```
alphabet = ["a", "b", "c", "d", "e", "f", "g", "h", "i", "j", "k", "l", "m", "n",
"o", "p", "q", "r", "s", "t", "u","v", "w", "x", "y", "z"]

FREQ = {"a":0.0815,"b":0.0144,"c":0.0276,"d":0.0379, "e":0.1311, "f":0.0292,
"g":0.0199, "h":0.0526, "i":0.0635, "j":0.0013, "k":0.0042, "l":0.0339, "m":0.0254,
"n":0.0710, "o":0.08, "p":0.0198, "q":0.0012, "r":0.0683, "s":0.0610, "t":0.1047,
"u":0.0246, "v":0.0092, "w":0.0154, "x":0.0017, "y":0.0198, "z":0.0008}

sorted_list_frequencies = []

all_letters = {}
```

basic IC (& dependencies)

```
# Occurence of letters in the text
def find_all_letters(text):
    all_letters.clear()
    clean_text = ''.join([i for i in text.lower() if i.isalpha()])
    for letter in clean_text:
        if letter not in all_letters:
            all_letters[letter] = 1
        elif letter in all_letters:
            all_letters[letter] += 1

# Index of coincidence
def basic_IC(text):
    chars = ''.join([i for i in text if i.isalpha()])
    find_all_letters(text)
    IC = 0.0
    for letter in all_letters:
        IC += all_letters[letter]*(all_letters[letter] - 1)
    IC/=(len(chars)*(len(chars)-1))
    return IC


basic_IC = 0.04933009857277119
```

```
all_letters = {'i': 3, 'g': 2, 'a': 1, 'u': 2, 'c': 2, 'm': 2, 'd': 1, 'p': 1, 's':
1, 'h': 1, 'b': 1, 'x': 2, 'w': 2, 'o': 1, 't': 3, 'z': 2, 'k': 1, 'j': 2}
```

Advanced IC + key length analysis

```
# General analysis of cipher under different shifts
def complete_frequency_analysis(text):
    # FInd the frequency for each letter
    frequency_analysis(text)
    # Correlation of frequencies
    correlations=[]
    for shift in range(26):
        print(str(shift) +" : ",end='')
        correlations.append(correlation(shift,text))



# This is trying to find the right shift by comparing
def correlation(shift,text):
    somme = 0.0
    index_of_all_letters = []
    for letter in all_letters:
        index_of_all_letters.append(alphabet.index(letter.lower()))
    for c in index_of_all_letters:
        # f(c) -> frequency of c in ciphertext
        # p(c-letter) -> regular frequency of character  c if it was shifted to the
left
        somme += frequency(c,text)*p((c-shift)%26)
    return somme


# frequency of a letter in the english dictionary
def p(letter):
    return FREQ[alphabet[letter]]


# frequency of character given in the ciphertext
def frequency(index_char,text):
    result = 0
    for freq in sorted_list_frequencies:
        if freq[0] == alphabet[index_char]:
            result=float(freq[1])
    return result

# 2nd IC method
def advanced_IC(text):
    result = basic_IC(text)*26
    return result

# Find the key length and the possible key
def find_key_length(text):
    chars = ''.join([i for i in text.lower() if i.isalpha()])
    # test key lengths from 1 to 10
    for length in range(1,11):
        column_strings = ["" for x in range(length)]
```

```
        text_divided = []
        division = ""
        for index in range(1,len(chars)+1):
            division+= chars[index-1]
            if index%length==0:
                text_divided.append(division)
                division=""
        for row in text_divided:
            for char in range(len(row)):
                column_strings[char] += row[char]
        print("=================")
        print("key length: "+str(length))
        delta_bar_IC = 0.0
        for i in column_strings:
            delta_bar_IC+= advanced_IC(i)
            # After finding the most likely key lengths let's look for the possible shifts for each column
            #complete_frequency_analysis(i)
        print(delta_bar_IC/length)
```

First key length analysis

```
key length analysis:
=================
key length: 1
1.2825825628920509
=================
key length: 2
1.2718763796909491
=================
key length: 3
1.8586138613861387
=================
key length: 4
1.2765765765765766
=================
key length: 5
1.3396610169491525
=================
key length: 6
1.8854421768707486
=================
key length: 7
1.205189052365132
=================
key length: 8
1.3956456456456456
=================
key length: 9
1.9368686868686866
=================
key length: 10
1.2551724137931033
```

2nd key length analysis with the correlation attack

```
key length: 3

IC of column 1: 0.053465346534653464

0 : 0.045788499999999996
1 : 0.0374296
2 : 0.043928
3 : 0.040039700000000004
4 : 0.050446300000000006
5 : 0.0346216999999999
6 : 0.033943
7 : 0.030101299999999994
8 : 0.0375186
9 : 0.035912099999999995
10 : 0.03456220000000001
11 : 0.0450818999999999
12 : 0.032365899999999996
13 : 0.0359881
14 : 0.0395842
15 : 0.0636033
16 : 0.0399002
17 : 0.0368451
18 : 0.03192130000000001
19 : 0.0400117999999999
20 : 0.035186300000000004
21 : 0.035151600000000005
22 : 0.03570160000000001
23 : 0.032398
24 : 0.0373939
25 : 0.0365758

IC of column 2: 0.0805940594059406

0 : 0.031167499999999997
1 : 0.0337802
2 : 0.0356302
3 : 0.0358174
4 : 0.04827560000000001
5 : 0.0400799
6 : 0.047865099999999994
7 : 0.0374371
8 : 0.04926860000000001
9 : 0.038984399999999995
10 : 0.0363458
11 : 0.026145199999999993
12 : 0.03791790000000001
13 : 0.032982099999999986
14 : 0.0321476
15 : 0.0414209
16 : 0.034012299999999995
17 : 0.0338049
18 : 0.038573800000000005
19 : 0.07587880000000001
20 : 0.0402409
```

```
21 : 0.032704500000000004
22 : 0.0262082
23 : 0.04633119999999999
24 : 0.03252409999999999
25 : 0.03545580000000001

IC of column 3: 0.0803960396039604

0 : 0.0298379
1 : 0.033455700000000005
2 : 0.037268800000000005
3 : 0.030728300000000004
4 : 0.033394
5 : 0.04267420000000001
6 : 0.07303079999999999
7 : 0.0427971
8 : 0.0344483
9 : 0.0310768
10 : 0.045608300000000004
11 : 0.03387129999999999
12 : 0.0343209
13 : 0.026469499999999997
14 : 0.0299656
15 : 0.0334563
16 : 0.041918800000000006
17 : 0.0503147
18 : 0.038465799999999994
19 : 0.04234860000000001
20 : 0.042383500000000005
21 : 0.0528133
22 : 0.04302890000000001
23 : 0.03826219999999999
24 : 0.026046800000000002
25 : 0.033013600000000004
```

Appendix B

Creation of the key depending on the informed guess

```
def findKey(text):
    key1=""
    key2=""
    for i in range(len(text)):
        if i ==0:
            key1 = ord(text[i]) ^ ord('A')
        elif i == 1:
            key2 = ord(text[i]) ^ ord(' ')
    print("key1: "+str(chr(key1)))
    print("key2: "+str(chr(key2)))
```

Test of that key

```
def decrypt(key,text):
```

```
    plain=""
    for i in range(len(text)):
        if i%2==0:
            plain+=chr(ord(text[i]) ^ ord(key[0]))
        elif i%2==1:
            plain+=chr(ord(text[i]) ^ ord(key[1]))
    print(plain)
```

Appendix C

XOR key bruteforce to find possible alphabetical strings

```
def find_key(text):
    #ascii chr
    ll = [chr(i) for i in range(128)]
    list_=[]
    for key1 in ll:
        for key2 in ll:
            possible_answer = ""
            for index in range(len(text)):
                if index%2 == 0:
                    possible_answer+= chr(ord(text[index]) ^ ord(key1))
                elif index%2 == 1:
                    possible_answer+= chr(ord(text[index]) ^ ord(key2))
            list_.append(possible_answer)
    list_ = test_list(list_)
    cap = []
    low = []
    for i in list_:
        if i.isupper():
            cap.append(i)
        elif i.islower():
            low.append(i)
    print(cap)
```