

Exercises

1. (Encryption and Statistical Analysis) Though encryption is primarily designed to preserve confidentiality and integrity of data, the mechanism itself is vulnerable to brute force (statistical analysis). In other words, the more we see the encrypted data, the easier we can hack it. In this exercise, you are asked to crack the following cipher text. Please provide the decrypted result and explain your strategy in decrypting this text.

Cipher text

PRCSOFQX FP QDR AFOPQ CZSPR LA JFPALOQSKR. QDFP FP ZK LIU BROJZK MOLTROE.

- a. Count the frequency of letters. List the top three most frequent characters.

Ans Top three most frequent letters:

Letter: P, Frequency: 7

Letter: R, Frequency: 6

Letter: O, Frequency: 6

- b. Knowing that this is English, what are commonly used three-letter words and two-letter words. Does the knowledge give you a hint on cracking the given text?

Ans Three-Letter Words: **THE, AND, FOR, ARE, BUT** and **NOT**

Two-Letter Words: **OF, TO, IN, IT, IS, AM, AND, AT** and **ON**

การระบุตัวแหน่งของคำสั้นๆที่ใช้กันทั่วไปสามารถนำมาใช้เพื่อวิเคราะห์ข้อความเข้ารหัสได้ว่า
อาจมีคำเหล่านี้อยู่

- c. Cracking the given text. Measure the time that you have taken to crack this message.

Ans 20 mins

- d. Explain your process in hacking such messages.

Ans

1.) PRCSOFQX **FP** QDR AFOPQ CZSPR LA JFPALOQSKR. QDFP FP ZK LIU BROJZK MOLTROE.

จากข้อความเหล่านี้ คิดว่า FP น่าจะต้องเป็นคำกริยา จึงเดาว่าเป็น IS ซึ่งส่งผลต่อไปว่า

F -> I และ P->S

2.) **SRCSOIQX is QDR AFOsQ CZSsR LA JisALOQSKR. QDis is ZK LIU**
BROJZK MOLTROE.

หลังจากแทนที่ F และ P ด้วย I และ S จึงมาสังเกตที่ QDR และคิดว่าเป็นคำว่า The เพราะเมื่อแทน QDR ด้วย The จะหมายความว่า Q -> T, D -> H และ R -> E ซึ่งทำให้ QDis -> This

3.) **SeCSOitX is the AFoSt CZSse LA JisALotSKe. This is ZK LIU**
BeOJZK MOLTeOE.

หลังจากนั้นจึงสังเกต AFoSt และลองหาคำที่มี 5 พยางค์และลงท้ายด้วย st มา จึงคิดว่าอาจจะเป็น first ซึ่งทำให้ A -> F, F -> I และ O -> R

4.) **SeCSritX is the first CZSse Lf JisfLrtSKe. This is ZK LIU**
BerJZK MrLTerE.

หลังจากนั้นจึงสังเกต SeCSritX ซึ่งคิดว่าน่าจะเป็น Security และจะทำให้ C -> C, S -> U, X -> Y

5.) **Security is the first cZuse Lf JisfLrtuKe. This is ZK LIU**
BerJZK MrLTerE.

ต่อมาจึงสังเกต cZuse ซึ่งคิดว่าน่าจะเป็น cause เพราะเมื่อแทน Z -> A จะทำให้ ZK -> aK ซึ่งทำให้วิเคราะห์ต่อไปได้ว่า ZK น่าจะเป็น an, และจึงมาสังเกต Lf ซึ่งคิดว่าน่าจะเป็น of จากทั้งหมดนี้จึงส่งผลให้ Z -> A, K -> N และ L -> O

6.) **Security is the first cause of Jisfortune. This is an oIU**
BerJan MroTerE.

ต่อมาจึงสังเกต Jisfortune คิดว่าน่าจะเป็น Misfortune จากคำที่ Google doc auto correct มาให้ จึงส่งผลให้ J -> M และทำให้ BerJan -> Berman ทำให้คิดได้ว่า Berman น่าจะเป็น German ซึ่งทำให้ B -> G

7.) **Security is the first cause of misfortune. This is an oIU**
German MroTerE.



X

<https://twitter.com/uhaaazezubi/status>

:

Security is the first cause of misfortune. German Proverb :-D

6 ก.ย. 2558 — **Security is the first cause of misfortune.** German Proverb :-D. 2:25 AM · Sep 6, 2015.

หลังจาก search ข้อความ “**Security is the first cause of misfortune**” ใน Google ทำให้คิดว่า MrTerE -> proverb ซึ่งหลังจากนั้นทำให้คิดว่า อยู่ -> old ทำให้ได้ประโยชน์คือ

Security is the first cause of misfortune. This is an old

German proverb.

- e. If you know that the encryption scheme is based on Caesar(Monoalphabetic Substitution) that is commonly used by Caesar for sending messages to Cicero, does it allow you to crack it faster?

Ans crack ได้เร็วขึ้น เพราะเรารู้ได้ว่าแต่ละตัวอักษรใน Cipher text จะแสดงถึงตัวอักษรที่แท้จริงเพียงตัวเดียว

- f. Draw a cipher disc of the given text.

Ans 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

Z E C U R A B D F G H I J K L M N O P Q S T V W X Y

- g. Create a simple python program for cracking the Caesar cipher text using brute force attack. Explain the design and demonstrate your software. (You may use an English dictionary for validating results.)

```
# import nltk
# nltk.download('words')
from nltk.corpus import words

from itertools import permutations

encrypt_text = 'PRCSOFOX FP QDR AFOPQ CZSPR LA JFPALOQSKR.'
encrypt_text_list = encrypt_text.split(" ")
alphabet = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
c = 'YODABCDEFHIJKLMNOPQRSTUVWXYZ'

def is_valid_word(word):
    return word.lower() in words.words()

def decrypt_text(key, text):
    decrypted_text = ''
    stop_word = ''
    for char in text:
        if char == ' ' or char == '.' or char == ',':
            stop_word = char
            continue
        index = (ord(char) - 65 + key) % 26
        decrypted_text += alphabet[index]
    return decrypted_text
```

```

    decrypted_text += alphabet[key.index(char)]


    if is_valid_word(decrypted_text):
        return decrypted_text + stop_word
    else:
        return ''


def count_most_repetitive_word():
    count = {}
    for word in encrypt_text_list:
        if word in count:
            count[word] += 1
        else:
            count[word] = 1

    sorted_data = sorted(count.items(), key=lambda item: item[1],
reverse=True)
    return sorted_data[0][0]


def count_frequency(text):
    unique_char_count = {}

    for char in text:
        if char == ' ' or char == '.' or char == ',':
            continue
        if char in unique_char_count:
            unique_char_count[char] += 1
        else:
            unique_char_count[char] = 1
    return unique_char_count


def top_three_frequent_chars(unique_char_count):
    count = {}
    for char in unique_char_count:
        if unique_char_count[char] in count:
            count[unique_char_count[char]].append(char)
        else:
            count[unique_char_count[char]] = [char]

    sorted_count = sorted(count.items(), reverse=True)
    top_three = sorted_count[:3]
    return top_three

```

```

most_repetitive_word = count_most_repetitive_word() # FP --> may be 'F' =
i, p = s'

def check_possible(cipher):
    if alphabet[cipher.index(most_repetitive_word[0])] != 'I' or
alphabet[cipher.index(most_repetitive_word[1])] != 'S':
        return False

    most_english_char = ['E', 'T', 'A', 'O', 'I', 'N', 'S', 'H', 'R', 'D',
'L']

    a = top_three_frequent_chars(count_frequency(encrypt_text))
    for i in range(len(a)):
        data = a[i][1]
        for j in range(len(data)):
            if data[j] in most_repetitive_word:
                continue
            elif alphabet[cipher.index(data[j])] not in most_english_char:
                return False
    return True


plain_text = ''
ShouldStop = False
for key_length in range(0,26):
    possible_key_list = list(permutations(alphabet, key_length))
    for possible_key in possible_key_list:
        key = ''.join([char for char in possible_key])
        cipher = key

        for char in alphabet:
            if char not in key:
                cipher += char

        c = check_possible(cipher)
        if c == False:
            continue

        for word in encrypt_text_list:
            ans = decrypt_text(cipher, word)
            if ans == '':
                plain_text = ''
                break
            else:
                plain_text += ans + " "

```

```

        ShouldStop = True
        if ShouldStop == True:
            break
        if ShouldStop == True:
            break

print("plain_text: ", plain_text)

```

● peeralis@sMacBookAir Activity04 % /usr/local/bin/python3 "/Users/peeralis/Documents/Compute
plain_text: SECURITY IS THE FIRST CAUSE OF MISFORTUNE. THIS IS AN OLD GERMAN PROVERB.

2.(Symmetric Encryption) Vigenère is a complex version of the Caesar cipher. It is a polyalphabetic substitution.

a.) Explain how it can be used to cipher data.

Ans Vigenère cipher เป็นการเข้ารหัสแบบพหุอักษร (polyalphabetic substitution) ที่ ซับซ้อนกว่าการเข้ารหัสแบบ Caesar โดยใช้คีย์เวิร์ดในการกำหนดการเลื่อนตัวอักษรแต่ละตัว ในข้อความ

ขั้นตอนการเข้ารหัส:

1. เลือกคีย์เวิร์ด: เช่น คำว่า KEY หากข้อความคือ HELLOWORLD คีย์เวิร์ดจะถูกทำซ้ำ เป็น KEYKEYKEYK
2. เลื่อนตัวอักษรตามคีย์เวิร์ด: แต่ละตัวอักษรในข้อความจะถูกเลื่อนตามตัวอักษรใน คีย์เวิร์ด ทำให้การเข้ารหัสแต่ละตำแหน่งต่างกัน

ผลลัพธ์ที่ได้จะเป็นข้อความที่เข้ารหัส (ciphertext) ซึ่งขั้นตอนและปลอดภัยกว่า Caesar cipher

b.) If a key is the word “CAT”, please analyze the level of security provided by Vigenère compared to that of the Caesar cipher.

Ans Vigenère cipher ที่ใช้คีย์คำว่า "CAT" ปลอดภัยกว่า Caesar cipher เพราะการเลื่อนตัว อักษรจะเปลี่ยนไปตามตัวอักษรในคีย์ เช่น หากข้อความคือ "HELLO" และใช้คีย์ "CAT" จะได้ ว่า ตัวอักษร H ถูกเลื่อนตาม C (เลื่อน 2 ตำแหน่งเป็น J) ตัวอักษร E ถูกเลื่อนตาม A และตัว อักษร L ถูกเลื่อนตาม T (เลื่อน 19 ตำแหน่งเป็น E) ซึ่งต่างจาก Caesar cipher ที่ทุกตัวอักษร จะถูกเลื่อนเท่ากันหมด ทำให้การวิเคราะห์ความถี่ของตัวอักษรทำได้ง่ายกว่า

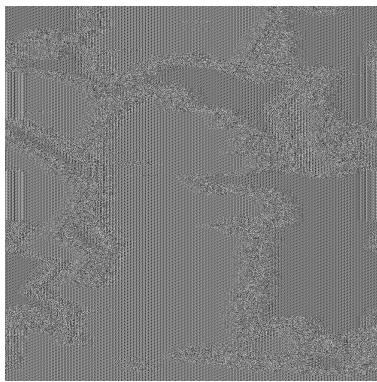
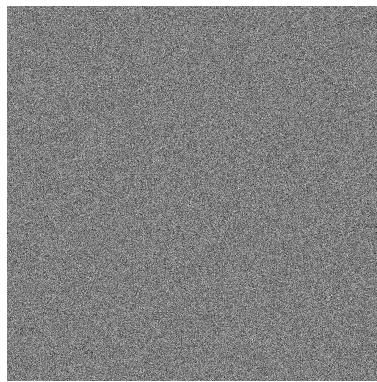
c.) Create a python program for ciphering data using Vigenère

```
1  def vigenere_cipher(text, key, mode='encrypt'):
2
3      def shift_letter(letter, shift):
4          if letter.isalpha():
5              shift = shift % 26
6              start = ord('A') if letter.isupper() else ord('a')
7              return chr(start + (ord(letter) - start + shift) % 26)
8          return letter
9
10     def get_shifts(key):
11         return [ord(char.upper()) - ord('A') for char in key]
12
13     key_shifts = get_shifts(key)
14     key_length = len(key_shifts)
15     result = []
16
17     for i, char in enumerate(text):
18         shift = key_shifts[i % key_length]
19         if mode == 'decrypt':
20             shift = -shift
21         result.append(shift_letter(char, shift))
22
23     return ''.join(result)
24
25 # Example usage
26 if __name__ == "__main__":
27     plaintext = "HELLO WORLD"
28     key = "CAT"
29
30     # Encrypt
31     encrypted = vigenere_cipher(plaintext, key, mode='encrypt')
32     print(f"Encrypted: {encrypted}")
33
34     # Decrypt
35     decrypted = vigenere_cipher(encrypted, key, mode='decrypt')
36     print(f"Decrypted: {decrypted}")
37
```

PROBLEMS OUTPUT TERMINAL PORTS GITLENS COMMENTS DEBUG CONSOLE

● peeralis@sMacBookAir Activity04 % /usr/local/bin/python3 "/Users/peeralis/Desktop/vigenere_cipher.py"
Encrypted: JEENO YOKND
Decrypted: HELLO WORLD

3. (Mode in Block Cipher) Block Cipher is designed to have more randomness in a block. However, an individual block still utilizes the same key. Thus, it is recommended to use a cipher mode with an initial vector, chaining or feedback between blocks. This exercise will show you the weakness of Electronic Code Book

Original	AES-256-ECB	AES-256-CBC
		

Ans การเข้ารหัสแบบ ECB มีข้อบกพร่องในการเข้ารหัส เพราะบล็อกที่เหมือนกันจะได้รับการเข้ารหัสเป็นผลลัพธ์ที่เหมือนกัน ทำให้สามารถเห็นลวดลายเดิมในข้อมูลที่เข้ารหัสได้ ในขณะที่ CBC ใช้ค่าเริ่มต้น (IV) และการเชื่อมโยงระหว่างบล็อก ทำให้ข้อมูลที่เหมือนกันจะได้รับการเข้ารหัสเป็นผลลัพธ์ที่แตกต่างกันและไม่สามารถเห็นลักษณะเดิมได้ ซึ่งช่วยเพิ่มความปลอดภัยให้กับข้อมูลมากขึ้นเมื่อเปรียบเทียบกับ ECB

4. (Encryption Protocol - Digital Signature)

- a. Measure the performance of a hash function (sha1), RC4, Blowfish and DSA.
Outline your experimental design.
(Please use OpenSSL for your measurement)

sha1:

```
peeralis@sMacBookAir ~ % openssl speed sha1
Doing sha1 for 3s on 16 size blocks: 25184294 sha1's in 2.99s
Doing sha1 for 3s on 64 size blocks: 15295019 sha1's in 2.99s
Doing sha1 for 3s on 256 size blocks: 6583519 sha1's in 2.99s
Doing sha1 for 3s on 1024 size blocks: 2013013 sha1's in 2.98s
Doing sha1 for 3s on 8192 size blocks: 271486 sha1's in 2.99s
```

RC4:

```
[peeralis@sMacBookAir ~ % openssl speed rc4
Doing rc4 for 3s on 16 size blocks: 207741029 rc4's in 2.99s
Doing rc4 for 3s on 64 size blocks: 61727297 rc4's in 2.99s
Doing rc4 for 3s on 256 size blocks: 16142710 rc4's in 2.99s
Doing rc4 for 3s on 1024 size blocks: 4051817 rc4's in 2.98s
Doing rc4 for 3s on 8192 size blocks: 510706 rc4's in 2.99s
```

Blowfish:

```
[peeralis@sMacBookAir ~ % openssl speed blowfish
Doing blowfish cbc for 3s on 16 size blocks: 25327420 blowfish cbc's in 2.99s
Doing blowfish cbc for 3s on 64 size blocks: 6584669 blowfish cbc's in 2.99s
Doing blowfish cbc for 3s on 256 size blocks: 1659405 blowfish cbc's in 2.98s
Doing blowfish cbc for 3s on 1024 size blocks: 415209 blowfish cbc's in 2.98s
Doing blowfish cbc for 3s on 8192 size blocks: 51888 blowfish cbc's in 2.98s
```

DSA:

```
[peeralis@sMacBookAir ~ % openssl speed dsa
Doing 512 bit sign dsa's for 10s: 158686 512 bit DSA signs in 9.94s
Doing 512 bit verify dsa's for 10s: 184249 512 bit DSA verify in 9.94s
Doing 1024 bit sign dsa's for 10s: 65848 1024 bit DSA signs in 9.95s
Doing 1024 bit verify dsa's for 10s: 65128 1024 bit DSA verify in 9.89s
Doing 2048 bit sign dsa's for 10s: 22025 2048 bit DSA signs in 9.95s
Doing 2048 bit verify dsa's for 10s: 20820 2048 bit DSA verify in 9.98s
```

- b. Comparing performance and security provided by each method.

อัลกอริธึม	ประสิทธิภาพ	ความปลอดภัย	สรุป
SHA-1	สูง	พบปัญหาการชนกัน (collision) ซึ่งสามารถมีสองข้อความที่แตกต่างกันให้คำแนะนำเหมือนกัน	ประสิทธิภาพสูงแต่ล้าสมัย และไม่ปลอดภัย
RC4	สูงมาก	มีช่องโหว่ด้านความปลอดภัย เช่น การโจมตีที่ทำให้เดาข้อมูลบางประเภทได้ง่ายขึ้น	รวดเร็วที่สุดแต่ไม่ปลอดภัย ไม่ควรใช้ในระบบการเข้ารหัสสมัยใหม่

Blowfish	ปานกลาง	มีความปลอดภัยที่ดีหากใช้ขนาดคีย์ที่เหมาะสม แต่ล้าสมัยเมื่อเปรียบเทียบกับมาตรฐานใหม่กว่า	ประสิทธิภาพและความปลอดภัยปานกลาง
DSA	ปานกลางค่อนไปทางต่ำ	ให้ความปลอดภัยดีเมื่อใช้ขนาดคีย์ที่ใหญ่ พอก แต่มีข้อจำกัดด้านประสิทธิภาพ	ประสิทธิภาพต่ำที่สุด ปลอดภัยหากใช้ขนาดคีย์ที่ใหญ่

- c. Explain the mechanism underlying Digital Signature. How does it combine the strength and weakness of each encryption scheme?

Ans ลายมือชื่อดิจิทัล (Digital Signature) ใช้ในการยืนยันความถูกต้องและความสมบูรณ์ของข้อมูล โดยเริ่มจากการแฮชข้อมูลด้วยฟังก์ชันแฮช (เช่น SHA-256) และใช้คีย์ส่วนตัว (Private Key) เข้ารหัสค่าแฮชเพื่อสร้างลายมือชื่อดิจิทัล ในการตรวจสอบ ลายมือชื่อจะถูกถอดรหัสด้วยคีย์สาธารณะ (Public Key) และเปรียบเทียบกับค่าแฮชที่สร้างใหม่จากข้อมูล หากตรงกันแสดงว่าข้อมูลไม่เปลี่ยนแปลงและลายมือชื่อเป็นของจริง การลงลายมือชื่อดิจิทัลรวมความแข็งแกร่งของการเข้ารหัสแบบไม่สมมาตรและฟังก์ชันแฮชเพื่อความปลอดภัยสูง แต่ต้องระวังจุดอ่อนของแต่ละเทคนิคด้วย