Activity IV - Fundamental of Cryptography

By Saenyakorn Siangsanoh 6232035721

Overviews

In this activity, we will learn the basics of encryption. There are 3 exercises in this activity. Each exercise is designed to let you learn the concepts of cryptography.

We will need:

- Imagemagick
- OpenSSL
- One of your favorite programming languages.

You are welcome to do this exercise with any programming language. If you have no preference, use python.

Exercises

(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.
```

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

In [2]:
```
text = "PRCSOFQX FP QDR AFOPQ CZSPR LA JFPALOQSKR. QDFP FP ZK LIU BROJZK MOL
f = dict()

def find_max_3(f):
    for i in range(3):
        max_key = max(f, key=f.get)
        print(max_key, f[max_key])
        f.pop(max_key)

for c in text:
    if c in f:
        f[c] += 1
    else:
        f[c] = 1

f.pop(' ')
```

```
print("Top 3 most frequent characters:")
find_max_3(f)
```

```
Top 3 most frequent characters:
P 7
R 6
O 6
```

1. 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?

Answer

ใช่ เพราะในตอนที่ลอง crack ครั้งแรก การ assume ว่า `FP` เป็น `is` และ `QDR` เป็น `the` ทำให้มองเห็นความสัมพันธ์ระหว่างตัวอักษร และทำให้เดาตัวอักษรถัดไปได้มากขึ้น

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

Answer

จากที่ได้ลอง ๆ ดูพบว่าตัวเองใช้เวลาในการเดา ประมาณ 30 นาที ซึ่งสามารถเดาได้ดังนี้

In [5]:
```
d = dict()
d['F'] = 'i'
d['P'] = 's'
d['Q'] = 't'
d['D'] = 'h'
d['R'] = 'e'
d['Z'] = 'a'
d['K'] = 'n'
d['C'] = 'c'
d['S'] = 'u'
d['O'] = 'r'
d['X'] = 'y'
d['A'] = 'f'
d['L'] = 'o'
d['J'] = 'm'
d['B'] = 'g'
d['I'] = 'l'
d['U'] = 'd'
d['E'] = 'b'
d['M'] = 'p'
d['T'] = 'v'

for c in text:
    if c in d:
        print(d[c], end='')
    else:
        print(c, end='')
```

```
security is the first cause of misfortune. this is an old german proverb.
```

1. Explain your process in hacking such messages.

Answer

จากที่ได้อธิบายไปในข้อ 2 ผมเริ่มเดาจาก คำที่มี 2 ตัวก่อน จากนั้น 3 ตัว และคำถัดจาก `the` มีความใกล้เคียงกับคำว่า `first` และ context ได้ เลยลองใส่ดูแล้วพบว่าคำแรกสุดใกล้เคียงคำว่า security จากนั้นก็เดาประโยคหลังต่อไป ตามคำที่นึกออกและคิดว่าใกล้เคียง รวมถึงใช้เป็นอักษรที่ยังไม่เคยเดาไปแล้ว

1. 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

Answer

ง่ายขึ้นอย่างมากเพราะจะได้ไม่ต้องมานั่งเดาตั้งแต่เริ่มว่า text นี้ใช้ encryption ท่าไหน

1. Draw a cipher disc of the given text.

In [12]:
```
p = list(d.items())
p.sort(key=lambda x: x[1])
top = []
bottom = []
for a, b in p:
  top.append(b)
  bottom.append(a)
print("".join(top))
print("".join(bottom))
```

```
abcdefghilmnoprstuvy
ZECURABDFIJKLMOPQSTX
```

1. 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.)

In [144…
```
import requests
import re

# Prepare dictionary
dictionary = requests.get("https://raw.githubusercontent.com/dwyl/english-wo
# Sample dictionary items
print(dictionary[100:120])
# Analyze dictionary
print("Length 2: ", len(list(filter(lambda x: (len(x) == 2), dictionary))))
print("Length 3: ", len(list(filter(lambda x: (len(x) == 3), dictionary))))
print("Length 4: ", len(list(filter(lambda x: (len(x) == 4), dictionary))))
print("Length 5: ", len(list(filter(lambda x: (len(x) == 5), dictionary))))
print("Length 6: ", len(list(filter(lambda x: (len(x) == 6), dictionary))))
print("Length 7: ", len(list(filter(lambda x: (len(x) == 7), dictionary))))
```

```
['aals', 'Aalst', 'Aalto', 'AAM', 'AAMSI', 'Aandahl', 'A-and-R', 'Aani', 'AA
O', 'AAP', 'AAPSS', 'Aaqbiye', 'Aar', 'Aara', 'Aarau', 'AARC', 'aardvark',
'aardvarks', 'aardwolf', 'aardwolves']
Length 2:  637
Length 3:  4711
Length 4:  11171
Length 5:  22950
Length 6:  39518
Length 7:  52093
```

In [145…

```
# Prepare cipher text
original_words = "PRCSOFQX FP QDR AFOPQ CZSPR LA JFPALOQSKR. QDFP FP ZK LIU

# Rearrange words
def reorder_words(words: list):
  score = dict()
  freq = dict()
  length = dict()
  # Unique words
  unique_words = list(set(words))
  # Measure character frequency
  for word in unique_words:
    for c in word:
      if c in freq:
        freq[c] += 1
      else:
        freq[c] = 1
  # Measure word score
  for word in unique_words:
    s = 0
    for c in word: s += freq[c]
    score[word] = s
  # Sort by length and score
  unique_words.sort(key=lambda x: -len(x)*score[x])
  for word in unique_words:
    print(word, len(word), score[word], len(word) / score[word])
  return unique_words

words = reorder_words(original_words)
print(words)
```

```
['JFPALOQSKR', 'PRCSOFQX', 'MOLTROE', 'BROJZK', 'AFOPQ', 'CZSPR', 'QDFP', 'Q
DR', 'FP', 'LIU', 'LA', 'ZK']
```

In [148…

```
def translate(d: dict, word: str) -> str:
    return "".join([d[c] if c in d else c for c in word])

def transform_to_regex(word: str) -> str:
    regex = "".join("." if c.isupper() else c for c in word])
    return f'^{regex}$'

def find_word_in_english_dictiory(regex: str, dictionary: list) -> list:
    r = re.compile(regex)
    return list(filter(r.match, dictionary))

def filter_words_with_dict_value(dv: list, words: list) -> list:
    results = []
    for word in words:
        if not any([c in dv for c in word]) and word.isalpha():
            results.append(word)
    return results

def store_translation(d: dict, word: str, translation: str) -> bool:
    for i in range(len(word)):
        if word[i].isupper():
            if translation[i] in d.values():
                return False
            d[word[i]] = translation[i]
    return True

def decrypt(d: dict, words: list, dictionary: list):
    if len(words) == 0:
        print("Found solution: ", d)
        return [d]
```

```
    top_word = words[0]
    translate_word = translate(d, top_word)
    if translate_word.islower():
        return decrypt(d, words[1:], dictionary)
    regex = transform_to_regex(translate_word)
    match_words = find_word_in_english_dictiory(regex, dictionary)
    # Filter out words that the chracaters are in the 'd'
    dc = [c for c in translate_word if c.islower()]
    dv = [v for v in list(d.values()) if v not in dc]
    match_words = filter_words_with_dict_value(dv, match_words)
    # Debug print
    join_match_words = ", ".join([x.lower() for x in match_words[:8]])
    print(f"Match {len(match_words)} for {top_word} ({regex}) -> {join_match
    print("Translate: ", translate(d, " ".join(original_words)))
    results = []
    for w in match_words:
        d2 = d.copy()
        if not store_translation(d2, translate_word, w.lower()):
            continue
        rs = decrypt(d2, words[1:], dictionary)
        if len(rs) > 0:
            results += rs
    return results

# Run decryption
stupid_dictionary = ["aa", "ab", "security", "is", "the", "first", "cause",
results = decrypt(dict(), words, dictionary)
print(results)
```

```
Match 48590 for JFPALOQSKR (^.........$) -> aardwolves, aaronsburg, abacter
ial, abalienate, abandoners, abandoning, abannition, abaptiston
Translate:  PRCSOFQX FP QDR AFOPQ CZSPR LA JFPALOQSKR QDFP FP ZK LIU BROJZK
MOLTROE
Match 0 for PRCSOFQX (^ds.otbi.$) ->
Translate:  dsCotbiX bd iDs ubtdi CZods cu abductions iDbd bd Zn cIU BstaZn
MtcTstE
Match 0 for PRCSOFQX (^ds.rtbo.$) ->
Translate:  dsCrtboX bd oDs ubtdo CZrds cu abductores oDbd bd Ze cIU BstaZe
MtcTstE
Match 0 for PRCSOFQX (^js.otbi.$) ->
Translate:  jsCotbiX bj iDs ebtji CZojs ce abjections iDbj bj Zn cIU BstaZn
MtcTstE
Match 0 for PRCSOFQX (^je.icbt.$) ->
Translate:  jeCicbtX bj tDe ubcjt CZije nu abjunctive tDbj bj Zv nIU BecaZv
McnTecE
Match 0 for PRCSOFQX (^oy.eibv.$) ->
Translate:  oyCeibvX bo vDy rbiov CZeoy tr abortively vDbo bo Zl tIU ByiaZl
MitTyiE
Match 0 for PRCSOFQX (^rt.egbm.$) ->
Translate:  rtCegbmX br mDt ibgrm CZert di abridgment mDbr br Zn dIU BtgaZn
MgdTtgE
Match 0 for PRCSOFQX (^ry.dtbe.$) ->
Translate:  ryCdtbeX br eDy ubtre CZdry pu abruptedly eDbr br Zl pIU BytaZl
MtpTytE
Match 0 for PRCSOFQX (^se.otbh.$) ->
Translate:  seCotbhX bs hDe ibtsh CZose ni absinthole hDbs bs Zl nIU BetaZl
MtnTetE
Match 0 for PRCSOFQX (^se.iubt.$) ->
Translate:  seCiubtX bs tDe obust CZise lo absolutive tDbs bs Zv lIU BeuaZv
MulTeuE
Match 0 for PRCSOFQX (^se.iubt.$) ->
Translate:  seCiubtX bs tDe obust CZise lo absolutize tDbs bs Zz lIU BeuaZz
MulTeuE
Match 0 for PRCSOFQX (^se.ipbt.$) ->
Translate:  seCipbtX bs tDe obpst CZise ro absorptive tDbs bs Zv rIU BepaZv
MprTepE
Match 0 for PRCSOFQX (^sn.ipbt.$) ->
Translate:  snCipbtX bs tDn ubpst CZisn mu absumption tDbs bs Zo mIU BnpaZo
MpmTnpE
Match 0 for PRCSOFQX (^en.rpcy.$) ->
Translate:  enCrpcyX ce yDn tcpey CZren ot acetopyrin yDce ce Zi oIU BnpaZi
MpoTnpE
Match 0 for PRCSOFQX (^hm.iocd.$) ->
Translate:  hmCiocdX ch dDm ecohd CZihm ne achenodium dDch ch Zu nIU BmoaZu
MonTmoE
Match 0 for PRCSOFQX (^he.idcr.$) ->
Translate:  heCidcrX ch rDe ocdhr CZihe no achondrite rDch ch Zt nIU BedaZt
MdnTedE
Match 0 for PRCSOFQX (^iy.tmce.$) ->
Translate:  iyCtmceX ci eDy dcmie CZtiy od acidometry eDci ci Zr oIU BymaZr
MmoTymE
Match 0 for PRCSOFQX (^re.iycm.$) ->
Translate:  reCiycmX cr mDe ocyrm CZire no acronymize mDcr cr Zz nIU BeyaZz
MynTeyE
Match 0 for PRCSOFQX (^um.oicf.$) ->
Translate:  umCoicfX cu fDm lciuf CZoum el aculeiform fDcu cu Zr eIU BmiaZr
MieTmiE
Match 0 for PRCSOFQX (^es.rvdi.$) ->
Translate:  esCrvdiX de iDs ndvei CZres on adenovirus iDde de Zu oIU BsvaZu
MvoTsvE
Match 0 for PRCSOFQX (^je.icdt.$) ->
Translate:  jeCicdtX dj tDe udcjt CZije nu adjunctive tDdj dj Zv nIU BecaZv
McnTecE
```

```
Match 0 for PRCSOFQX (^nd.ahew.$) ->
Translate:  ndCahewX en wDd iehnw CZand ti zenithward wDen en Zr tIU BdhzZr
 MhtTdhE
Match 0 for PRCSOFQX (^ny.prea.$) ->
Translate:  nyCpreaX en aDy oerna CZpny go zenography aDen en Zh gIU ByrzZh
 MrgTyrE
Match 0 for PRCSOFQX (^ps.nrei.$) ->
Translate:  psCnreiX ep iDs herpi CZnps yh zephyrinus iDep ep Zu yIU BsrzZu
 MryTsrE
Match 0 for PRCSOFQX (^gh.nrya.$) ->
Translate:  ghCnryaX yg aDh oyrga CZngh bo zygobranch aDyg yg Zc bIU BhrzZc
 MrbThrE
Match 0 for PRCSOFQX (^gn.rtye.$) ->
Translate:  gnCrtyeX yg eDn oytge CZrgn po zygopteran eDyg yg Za pIU BntzZa
 MtpTntE
Match 0 for PRCSOFQX (^gd.rtye.$) ->
Translate:  gdCrtyeX yg eDd oytge CZrgd po zygopterid eDyg yg Zi pIU BdtzZi
 MtpTdtE
Match 0 for PRCSOFQX (^gs.rtye.$) ->
Translate:  gsCrtyeX yg eDs oytge CZrgs po zygopteris eDyg yg Zi pIU BstzZi
 MtpTstE
Match 0 for PRCSOFQX (^nh.agir.$) ->
Translate:  nhCagirX in rDh cignr CZanh oc zincograph rDin in Zp oIU BhgzZp
 MgoThgE
[{'J': 'm', 'F': 'i', 'P': 's', 'A': 'f', 'L': 'o', 'O': 'r', 'Q': 't', 'S':
'u', 'K': 'n', 'R': 'e', 'C': 'c', 'X': 'y', 'M': 'p', 'T': 'v', 'E': 'b',
 'B': 'd', 'Z': 'a', 'D': 'h', 'I': 'w', 'U': 'k'}, {'J': 'm', 'F': 'i',
 'P': 's', 'A': 'f', 'L': 'o', 'O': 'r', 'Q': 't', 'S': 'u', 'K': 'n', 'R':
 'e', 'C': 'c', 'X': 'y', 'M': 'p', 'T': 'v', 'E': 'b', 'B': 'd', 'Z': 'a',
 'D': 'h', 'I': 'w', 'U': 'l'}, {'J': 'm', 'F': 'i', 'P': 's', 'A': 'f',
 'L': 'o', 'O': 'r', 'Q': 't', 'S': 'u', 'K': 'n', 'R': 'e', 'C': 'c', 'X':
 'y', 'M': 'p', 'T': 'v', 'E': 'b', 'B': 'g', 'Z': 'a', 'D': 'h', 'I': 'w',
 'U': 'd'}, {'J': 'm', 'F': 'i', 'P': 's', 'A': 'f', 'L': 'o', 'O': 'r',
 'Q': 't', 'S': 'u', 'K': 'n', 'R': 'e', 'C': 'c', 'X': 'y', 'M': 'p', 'T':
 'v', 'E': 'b', 'B': 'g', 'Z': 'a', 'D': 'h', 'I': 'w', 'U': 'k'}, {'J':
 'm', 'F': 'i', 'P': 's', 'A': 'f', 'L': 'o', 'O': 'r', 'Q': 't', 'S': 'u',
 'K': 'n', 'R': 'e', 'C': 'c', 'X': 'y', 'M': 'p', 'T': 'v', 'E': 'b', 'B':
 'g', 'Z': 'a', 'D': 'h', 'I': 'w', 'U': 'l'}, {'J': 'm', 'F': 'i', 'P':
 's', 'A': 'f', 'L': 'o', 'O': 'r', 'Q': 't', 'S': 'u', 'K': 'n', 'R': 'e',
 'C': 'c', 'X': 'y', 'M': 'p', 'T': 'v', 'E': 'b', 'B': 'j', 'Z': 'a', 'D':
 'h', 'I': 'w', 'U': 'd'}, {'J': 'm', 'F': 'i', 'P': 's', 'A': 'f', 'L':
 'o', 'O': 'r', 'Q': 't', 'S': 'u', 'K': 'n', 'R': 'e', 'C': 'c', 'X': 'y',
 'M': 'p', 'T': 'v', 'E': 'b', 'B': 'j', 'Z': 'a', 'D': 'h', 'I': 'w', 'U':
 'k'}, {'J': 'm', 'F': 'i', 'P': 's', 'A': 'f', 'L': 'o', 'O': 'r', 'Q':
 't', 'S': 'u', 'K': 'n', 'R': 'e', 'C': 'c', 'X': 'y', 'M': 'p', 'T': 'v',
 'E': 'b', 'B': 'j', 'Z': 'a', 'D': 'h', 'I': 'w', 'U': 'l'}, {'J': 'm',
 'F': 'i', 'P': 's', 'A': 'f', 'L': 'o', 'O': 'r', 'Q': 't', 'S': 'u', 'K':
 'n', 'R': 'e', 'C': 'c', 'X': 'y', 'M': 'p', 'T': 'v', 'E': 'b', 'B': 'k',
 'Z': 'a', 'D': 'h', 'I': 'w', 'U': 'd'}, {'J': 'm', 'F': 'i', 'P': 's',
 'A': 'f', 'L': 'o', 'O': 'r', 'Q': 't', 'S': 'u', 'K': 'n', 'R': 'e', 'C':
 'c', 'X': 'y', 'M': 'p', 'T': 'v', 'E': 'b', 'B': 'k', 'Z': 'a', 'D': 'h',
 'I': 'w', 'U': 'l'}]
```

In [154…

```
for i, d in enumerate(results):
    print(f"Solution {i+1}: ", translate(d, " ".join(original_words)))
```

```
Solution 1:  security is the first cause of misfortune this is an owk derman
proverb
Solution 2:  security is the first cause of misfortune this is an owl derman
proverb
Solution 3:  security is the first cause of misfortune this is an owd german
proverb
Solution 4:  security is the first cause of misfortune this is an owk german
proverb
Solution 5:  security is the first cause of misfortune this is an owl german
proverb
Solution 6:  security is the first cause of misfortune this is an owd jerman
proverb
Solution 7:  security is the first cause of misfortune this is an owk jerman
proverb
Solution 8:  security is the first cause of misfortune this is an owl jerman
proverb
Solution 9:  security is the first cause of misfortune this is an owd kerman
proverb
Solution 10:  security is the first cause of misfortune this is an owl kerma
n proverb
```

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

1. Based on the Caesar cipher, explain how it can be used to cipher data.

Answer

สำหรับ Caesar cipher วิธีการเข้ารหัสก็คือ เราจะกำหนด key ขึ้นมาตัวหนึ่งก่อนที่รู้ทั้งผู้รับและผู้ส่ง สมมติว่าเป็น `CAT` จากนั้นทำการเขียน Caesar Disc เพื่อกำหนดว่าตัวอักษรนี้จะ map ไปเป็นตัวอักษรอะไร โดนเราจะต้องต้นที่ Key ก่อน จากนั้นตัวอัษรภาษาังกฤษที่เหลือจะต่อตามหลัง (ไม่เอาตัวที่เขียนไปแล้ว) ดังนี้

abcdefghijklmnopqrstuvwxyz

catbdefghijklmnopqrsuvwxyz

จากนั้นเราก็จะสามารถเข้ารหัสแและถอดรหัสได้เลย ตราบใดที่รู้ key

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

Answer

ในกรณี Caesar cipher ถ้าเรามี Key เป็น `CAT` จะพบว่าเราสามารถเดา encrypted message ได้ง่ายมาก เพราะถ้าเดา context และ vocab ได้ก็พอจะเดา key ออกได้เลย เพราะมัน map แบบ 1-1

แต่ในกรณีของ Vigenère จะพบว่า แม้ Key จะเป็น `CAT` แต่เวลา encrypt แต่ละตัวอักษรมันจะ map ไปอีกตัวอักษรไม่เหมือนกัน เช่น

`HELLO` -> `JEENO`

`EHOOL` -> `GHHQL`

ซึ่งทำให้เราเดาได้ยากกว่าเดิมมากว่า raw data จริง ๆ แล้วเป็นอะไร

1. Create a python program for ciphering data using Vigenère

In [4]:
```
# Create vigenere table
def create_vigenere_table() -> dict:
  vigenere = dict()
  for i in range(26):
    vigenere[chr(i+65)] = dict()
    for j in range(26):
      vigenere[chr(i+65)][chr(j+65)] = chr((i+j) % 26 + 65)
  return vigenere

# Print vigenere table
vigenere = create_vigenere_table()
for i in range(26):
  print(" ".join([vigenere[chr(i+65)][chr(j+65)] for j in range(26)]))
```

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

In [46]:
```
# Ciphering data
def ciphering(data: str, key: str):
  result = ""
  for i in range(len(data)):
    result += vigenere[data[i].upper()][key[i % len(key)]]
  return result


ciphering("HELLOWORLD", "CAT")
```

Out[46]:
```
'JEENOPQREF'
```

(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 mode which does not include any initial vector, chaining or feedback.

1. What does the result suggest about the mode of operation in block cipher? Please provide your analysis.

Answer

To chiper the image please use the following command:

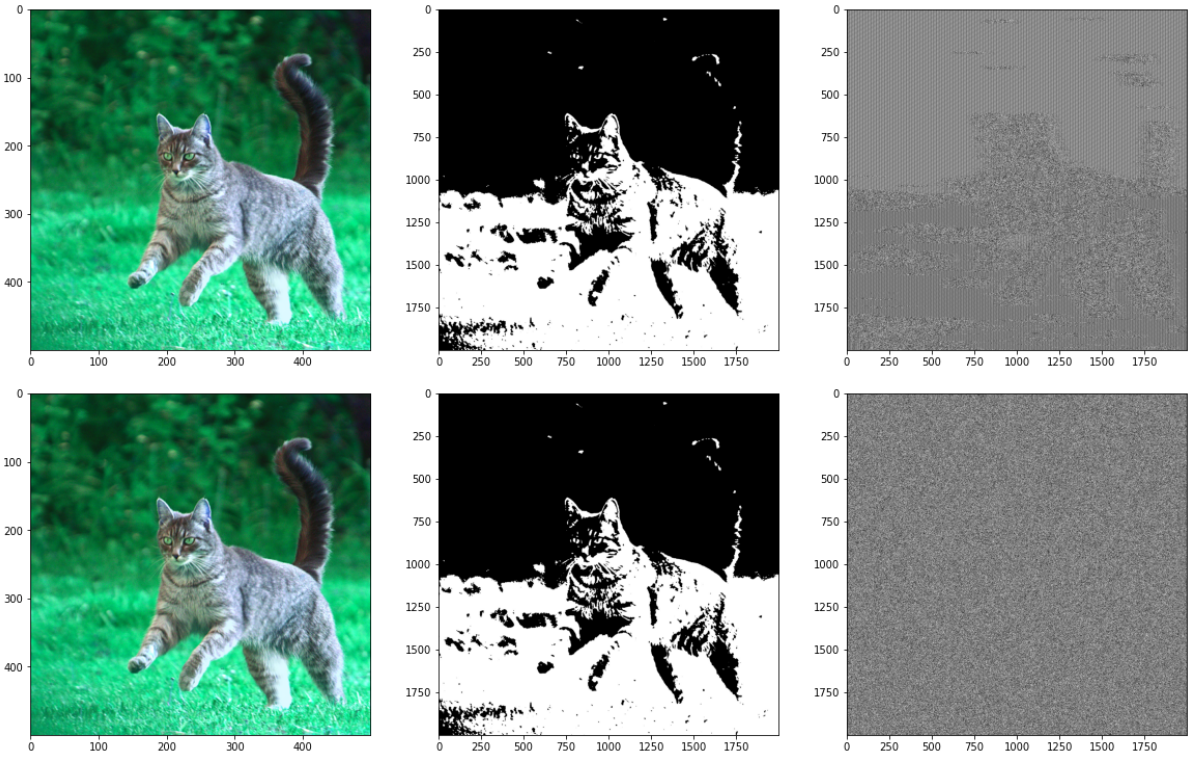
```
mkdir assets/<method>
convert assets/image.png -resize 2000x2000 assets/aes-256-
ecb/org.pbm
tail -n +2 assets/<method>/org.pbm > assets/<method>/org.x
openssl enc -<method> -in assets/<method>/org.x -nosalt -out
assets/<method>/enc.x
{ echo -n 'P4\n2000 2000\n'; cat assets/<method>/enc.x ; } >
assets/<method>/enc.pbm
```

จะพบว่าผลลัพธ์ที่ได้เป็นดังนี้

In [29]:
```
import matplotlib.pyplot as plt
import cv2


def plot_pbm(image_paths: list):
  # Plot row
  fig, axs = plt.subplots(1, len(image_paths), figsize=(20, 20))
  for i, image_path in enumerate(image_paths):
    image = cv2.imread(image_path)
    axs[i].imshow(image, cmap='gray')
  plt.show()

plot_pbm(["assets/image.png", "assets/aes-256-ecb/org.pbm", "assets/aes-256-
plot_pbm(["assets/image.png", "assets/aes-256-cbc/org.pbm" ,"assets/aes-256-
```



จากการทดลองโดยใช้ AES-256-ECB (ดังในแถวที่ 1) และใช้ AES-256-CBC (ดังในแถวที่สอง) พบว่าการ encrypt โดยใช้ AES-256-ECB จะได้ผลลัพธ์ที่เห็นเค้าลางของภาพเดิมอยู่หน่อย ๆ ซึ่งอาจ

จะทำให้คนร้ายเดาได้ว่าภาพต้นฉบับเป็นภาพอะไร (ในกรณีที่ encrpyt password ก็อาจจะทำให้ผู้ร้ายเดารหัสต้นฉบับได้)

ในขณะเดียวกันแบบ AES-256-CBC จะมองไม่ค่อยเห็นอะไรเลย ซึ่งทำให้ผผู้ร้ายเดาออกได้ยากกว่า

ดังนั้นแนะนำให้ใช้ AES-256-CBC มากกว่า AES-256-ECB

(Encryption Protocol - Digital Signature)

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

In [40]:
```
import os

# Generate large file with random data
def generate_large_file(file_path: str, size: int):
  with open(file_path, "wb") as f:
    f.write(os.urandom(size))

# Generate large file
generate_large_file("assets/large-file", 1024*1024*1024)
```

In [5]:
```
# Generate DSA parameters and key
!mkdir -p assets/dsa
!openssl dsaparam 1024 < /dev/random > assets/dsa/dsaparam.pem
!openssl gendsa assets/dsa/dsaparam.pem -out assets/dsa/dsa_priv.pem
!openssl dsa -in assets/dsa/dsa_priv.pem -pubout -out assets/dsa/dsa_pub.pem
!openssl dgst -sha1 -out assets/dsa/dsa_digest.sha1 assets/large-file
```

```
Generating DSA parameters, 1024 bit long prime
This could take some time
.........+..................+..............+.........+.+........+......+.
+............+.....+..+.........................+............++++++++++++
++++++++++++++++++++++++++++++++++++*
..........+....+..+..+...+++++++++++++++++++++++++++++++++++++++++++++++++
*
Generating DSA key, 1024 bits
read DSA key
writing DSA key
```

รัน command ดังนี้เพื่อวัดผลลัพธ์

```
for i in {1..10}
do
  time openssl dgst -sha1 -out /dev/null assets/large-file
done

for i in {1..10}
do
  time openssl enc -rc4 -nosalt -k -nopad -in assets/large-file -
out /dev/null
done

for i in {1..10}
do
  time openssl enc -bf -nosalt -k -nopad -in assets/large-file -
out /dev/null
done
```

```
for i in {1..10}
do
  time openssl dgst -dss1 -sign assets/dsa/dsa_priv.pem -out
/dev/null assets/large-file
done
```

ได้ผลลัพธ์ดังนี้

SHA1	RC4	Blowfish	DSA
2.012	2.432	7.734	1.728
1.412	2.029	7.38	1.641
1.454	2.074	7.384	1.677
1.446	2.167	7.473	1.684
1.417	2.083	7.991	1.428
1.452	2.061	7.556	1.411
1.6	2.025	7.874	1.55
1.449	2.034	7.92	1.607
1.437	2.026	7.473	1.476
1.571	2.012	7.453	1.424

1. Comparing performance and security provided by each method.

Answer

พบว่า

Method	Time(s)
sha1	1.525
rc4	2.0943
blowfish	7.6238
dsa	1.5626

ดังนั้น Method ที่ใช้เวลาช้าที่สุดคือ blowfish ซึ่งเป็น block cipher และมีความปลอดภัยสูง

ในขณะที่ sha1 จะเป็น method ที่เร็วที่สุด แต่มีความปลอดภัยต่ำ

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

Answer

SHA1 จะเป็น method ที่มีความซับซ้อนน้อยจึงทำให้มีความเร็วสูง แต่มีความปลอดภัยต่ำ เพราะมีขนาดแค่ 160 bit เท่านั้น ซึ่งมีความเสี่ยงที่จะถูกเดาออกได้ นอกจากนี้ยังมีโอกาสที่จะ hash แล้วไปซ้ำกับข้อมูลอื่น ๆ ได้

RC4 จะเป็น method ในลักษณะ stream cipher ซึ่งมีความซับซ้อนมากขึ้น และใช้ XOR operation ในการ encrypt ข้อมูล แต่ stream cipher ก็มีวิธีโจมตีได้หลายหลาย ซึ่งก็มีความเสี่ยงที่จะถูกเดาออกได้

Blowfish จะเป็น method ในลักษณะ block cipher ซึ่งมีความซับซ้อนมากขึ้นมาก และมีโอกาสโจมตีได้ยากขึ้น แต่ใช้เวลาในการ encrypt ข้อมูลนานขึ้น

DSA เป็น Digital Signature Algorithm ซึ่งก็คือการสร้าง key มาสองอันคือ public key และ private key จากนั้นเอาข้อมูลไป digest และ sign ด้วย key ที่สร้างขึ้น ซึ่งผลลัพธ์คือได้เวลาในการทำงานใกล้เคียงกับ SHA1 มาก