tp1_report

October 1, 2019

1 TP1 Report - Computer Security

Ning, Tien-Tso (Kense)
1 October 2019

```
In [1]: import numpy as np
        import math
```

2 Exercise 1: Modular Arithmetic

For n = 10, draw:
The addition table of (Zn,+)

```
In [2]: n = 10
        a = np.zeros((10,10)) #ten-by-ten table
        for i in range (0, 10):
            for j in range (0, 10):
                a[i][j] = (i+j)%n
        print(a)
```

```
[[0. 1. 2. 3. 4. 5. 6. 7. 8. 9.]
 [1. 2. 3. 4. 5. 6. 7. 8. 9. 0.]
 [2. 3. 4. 5. 6. 7. 8. 9. 0. 1.]
 [3. 4. 5. 6. 7. 8. 9. 0. 1. 2.]
 [4. 5. 6. 7. 8. 9. 0. 1. 2. 3.]
 [5. 6. 7. 8. 9. 0. 1. 2. 3. 4.]
 [6. 7. 8. 9. 0. 1. 2. 3. 4. 5.]
 [7. 8. 9. 0. 1. 2. 3. 4. 5. 6.]
 [8. 9. 0. 1. 2. 3. 4. 5. 6. 7.]
 [9. 0. 1. 2. 3. 4. 5. 6. 7. 8.]]
```

--
The multiplication table of (Zn, *)

```
In [3]: n = 10
        a = np.zeros((10,10)) #ten-by-ten table
```

```
        for i in range (0, 10):
            for j in range (0, 10):
                a[i][j] = (i*j)%n
        print(a)
```

```
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 2. 3. 4. 5. 6. 7. 8. 9.]
 [0. 2. 4. 6. 8. 0. 2. 4. 6. 8.]
 [0. 3. 6. 9. 2. 5. 8. 1. 4. 7.]
 [0. 4. 8. 2. 6. 0. 4. 8. 2. 6.]
 [0. 5. 0. 5. 0. 5. 0. 5. 0. 5.]
 [0. 6. 2. 8. 4. 0. 6. 2. 8. 4.]
 [0. 7. 4. 1. 8. 5. 2. 9. 6. 3.]
 [0. 8. 6. 4. 2. 0. 8. 6. 4. 2.]
 [0. 9. 8. 7. 6. 5. 4. 3. 2. 1.]]
```

--

The group of invertible elements (Z*n)

```
In [4]: invertibles = []
        for i in range (1,n):
            for j in range(1,n):
                if (i*j)%n == 1:
                    invertibles.append(i)
                    invertibles.append(j)
        invertibles = set(invertibles)
        print(sorted(invertibles))
```

```
[1, 3, 7, 9]
```

--

Euler's Totient function ϕ(n)

```
In [5]: totatives = []
        tfun = 0
        for i in range (1,n):
            if math.gcd(i,n)==1:
                totatives.append(i)
                tfun += 1
        totatives = set(totatives)
        print("Euler's Totient function of n :", tfun)
        print("Totatives: ", sorted(totatives))
```

```
Euler's Totient function of n : 4
Totatives:  [1, 3, 7, 9]
```

--
Compute:
[4+5]7 = [9]7 = [2]7 = 2
[11]7 - [17]7 = [11-17]7 = [-6]7 = -6
[11]7 * [17]7 = [11*17]7 = [187]7 = 5
[212741]8 = [23247]8 = [7]8 = 7
[-44]3 = [-2]3 = -2

3 Exercise 2: Caesar Cipher

Code the message "BONJOUR" with a key of 4.

```
In [6]: def caesar(message, k_ey):

            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".split(",")
            ch2int = {key: value for (value, key) in enumerate(alphabet)}
            int2ch = {key: value for (key, value) in enumerate(alphabet)}

            plain_text = message
            cipher_text = [0]*len(plain_text)

            for i in range(0,len(plain_text)): #encrypt
                cipher_text[i] = (ch2int[plain_text[i]]+k_ey)%26
                cipher_text[i] = int2ch[cipher_text[i]]

            return "".join(cipher_text)

        def un_caesar(message, k_ey):

            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".split(",")
            ch2int = {key: value for (value, key) in enumerate(alphabet)}
            int2ch = {key: value for (key, value) in enumerate(alphabet)}

            cipher_text = message
            plain_text = [0]*len(cipher_text)

            for i in range(0,len(cipher_text)): #decrypt
                plain_text[i] = abs((ch2int[cipher_text[i]]-k_ey))%26
                plain_text[i] = int2ch[plain_text[i]]

            return "".join(plain_text)

        print("Cipher Text is: ", caesar("bonjour", 4))
        print("Plain Text is: ", un_caesar(caesar("bonjour", 4), 4))
```

```
Cipher Text is:  fsrnsyv
Plain Text is:  bonjour
```

Since you cannot do a 0 shift, or n shifts since that brings you back to the same value, for an alphabet of X characters, you effectively have $X - 1$ keys.

An algorithm that can break the cipher would just shift the cipher text backwards consistently by an incrementing value K (key), assuming an alphabet of 26 characters, you would only need to do this 25 times.

Functions caesar and un_caesar are both defined above, and the output of the cipher text and plain text above confirms that the functions work as intended.

4 Exercise 3: Monoalphabetic Substitution

Give the complete key of "substitutionmonoalphabetique" and the original message for "BGYDKCNGSDMAIBHSJJAFRI".

```
In [7]: def create_key (phrase):
            phrase = phrase.lower() #lowercases your phrases
            your_key = {}
            roll_over = 97
            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".split()
            for ch in phrase:
                if ch in your_key:
                    pass
                else:
                    your_key[ch] = str(chr(roll_over))
                    roll_over += 1

            for x in alphabet:
                if x in your_key:
                    pass
                else:
                    your_key[x] = str(chr(roll_over))
                    roll_over += 1

            return your_key

        def monoalphabetic_substitution(message, key):
            output = []
            for ch in message:
                output.append(key[ch])

            return "".join(output)

        k_ey = create_key("substitutionmonoalphabetique")
        print("Key is: ", "".join(k_ey.keys()))
        print("Encrypting 'gomennewatashi': ", monoalphabetic_substitution("gomennewatashi", k
```

```
Key is:  subtionmalpheqcdfgjkrvwxyz
Encrypting 'gomennewatashi':  rfhmggmwidiale
```

Is it easy to break this code? Given that the plain-text characters and cipher-text characters are a deterministic exchange, breaking the code would just be a task of guessing the correct ciper-text exchange which can be done using frequency analysis. Frequency analysis helps because certain letters will appear more frequently than other characters, and certain characters will appear more likely if it follows a specific character (i.e: 'u' is very likely after the letter 'q' in the English language). Using this knowledge, you can guess which characters exchange with which characters in the cipher-text in order to obtain the plain-text.

5 Exercise 4: Vigenere Algorithm

Encode the message "JESUISUNPOKEMON" using the key "ABRA"

```
In [8]: message = "JESUISUNPOKEMON".lower()
        int_representation = []
        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".split(",")
        ch2int = {key: value for (value, key) in enumerate(alphabet)}
        int2ch = {key: value for (key, value) in enumerate(alphabet)}
        for ch in message:
            int_representation.append(ch2int[ch])
        print("Message before encryption: ", int_representation)
        k_ey = [ch2int['a'],ch2int['b'],ch2int['r'],ch2int['a']]
        for i in range(0, len(int_representation)):
            int_representation[i] = (int_representation[i] + k_ey[i%len(k_ey)-1])%26
        print("Message after encryption: ", int_representation)
        print("Message in cipher-text form: ", "".join([int2ch[x] for x in int_representation])
```

```
Message before encryption:  [9, 4, 18, 20, 8, 18, 20, 13, 15, 14, 10, 4, 12, 14, 13]
Message after encryption:  [9, 4, 19, 11, 8, 18, 21, 4, 15, 14, 11, 21, 12, 14, 14]
Message in cipher-text form:  jetlisvepolvmoo
```

```
In [9]: cipher_message = "JFJUITLNDSVSSFLR".lower()
        decrypt_int = []
        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".split(",")
        ch2int = {key: value for (value, key) in enumerate(alphabet)}
        int2ch = {key: value for (key, value) in enumerate(alphabet)}
        for ch in cipher_message:
            decrypt_int.append(ch2int[ch])
        print("Message before decryption: ", decrypt_int)
        k_ey = [ch2int['a'],ch2int['b'],ch2int['r'],ch2int['a']]
        for i in range(0, len(decrypt_int)):
            decrypt_int[i] = (decrypt_int[i] - k_ey[i%len(k_ey)-1])%26
        print("Message after decryption: ", decrypt_int)
        print("Message in plain-text form: ", "".join([int2ch[x] for x in decrypt_int]))
```

```
Message before decryption:  [9, 5, 9, 20, 8, 19, 11, 13, 3, 18, 21, 18, 18, 5, 11, 17]
Message after decryption:  [9, 5, 8, 3, 8, 19, 10, 22, 3, 18, 20, 1, 18, 5, 10, 0]
Message in plain-text form:  jfiditkwdsubsfka
```

Knowing that the key is much shorter than the message, it is still difficult to break the cipher because of the rotating nature of the key. The same plain-text characters will not deterministically be exchanged with cipher-text characters, and thus a frequency analysis does not provide information readily. If the key was the same length as the message, no rotation would occur, and you would have a caesar cipher/monoalphabetic substitution, which can be cracked easily due to frequency analysis.

In [ ]: