By-Varun Singhal(MT20099)
Sachin Kotiyal(MT20093)

CSE 540 Network Security
Assignment 1
Project 0

Question: Encryption & decryption using mono-alphabetic substitution of a pair of characters at a time, viz. <xy>, where x ε {A, B, C}, y ε {A, B, C}. Encryption uses a table consisting of 3x3 = 9 rows of tuples of the kind e.g. ABBD, AC BA, etc. The resulting ciphertext pair of characters is <pq>, where p ε {A, B, C}, q ε {A, B, C}. Then develop the software to launch a brute-force attack to discover the key. The plaintext should be long enough and “recognizable” as described above.

Solution Steps:

1. **Initialize a key.**
2. **Assume a 5 plaintext.**
3. **Add crc (checksum) after the plaintext.**
4. **Encrypt the 5 plaintext into ciphertext with the same key.**
5. **And decrypt the 5 ciphertext with the same key.**
6. **And then finally do BruteForce attack on the same 5 ciphertext after generating the 9! Keys.**

By-Varun Singhal(MT20099)
Sachin Kotiyal(MT20093)

- **Step1: Initialize a key.**

  Here we initialize the as a string that stores the rank of order of pair of <xy> where x ε {A, B, C}, y ε {A, B, C}.
  And the index of that string represents the lexecographical order of the pair of <xy> where x ε {A, B, C}, y ε {A, B, C}.
  It will easy for me to compute all permutation of a key using this string representation as a key of order pait <xy>

```
key = "345678012"  #its means #key = {'AA': "BA",
                                #   'AB': "BB",
                                #   'AC': "BC",
                                #   'BA': "CA",
                                #   'BB': "CB",
                                #   'BC': "CC",
                                #   'CA': "AA",
                                #   'CB': "AB",
                                #   'CC': "AC"}

# Rank = {'AA': 0,
#         'AB': 1,
#         'AC': 2,
#         'BA': 3,
#         'BB': 4,
#         'BC': 5,
#         'CA': 6,
#         'CB': 7,
#         'CC': 8}
```

- **Step2: Assume a 5 plaintext.**

Here we take the five plaintext which contain the character set {A,B,C} only.

```
Original Plaintext :

AAABACBABBBCCACBCC
AAABACCACBCCBABBBC
CACBCCBABBBCAAABAC
CCCBCAAAABACBABCBB
BCBBBAAAABACCACCCB
```

- ## Step3: Add crc (checksum) after the plaintext.

First we convert plaintext into the bytes and by using the help of zlib.crc32 calculate the checksum for every plaintext and append with the plaintext by mediator ~ operator.

Using this Fucntion we calculate the crc Checksum given after that we append this into the Plaintext with '~' charcharter:

```
#hash function which return checksum.
def hash_Function(plaintext):
  plaintext_bytes = bytes(plaintext, 'ascii')
  checksum = zlib.crc32(plaintext_bytes)
  checksum = str(checksum)
  return checksum
```

Output:

```
Plaintext with add checksum :

AAABACBABBBCCACBCC~1261564828
AAABACCACBCCBABBBC~2567393746
CACBCCBABBBCAAABAC~4259082963
CCCBCAAAABACBABCBB~2454973805
BCBBBAAAABACCACCCB~614313706
```

- ## Step4: Encrypt the 5 plaintext into ciphertext with the same key.

Using the **EncryptMessage(plaintext, key) function** i will convert the the text before the '~' operator into cipher text using that key value passed and then return the text contain converted ciphertext + one tilled operator i.e '~' with crc checksum value.

By-Varun Singhal(MT20099)
Sachin Kotiyal(MT20093)

Function of EncryptMessage(plaintext, key) :

```
# Encrption Message
def encrptMessage(plaintext, key):

  table1 = {'AA': 0,
            'AB': 1,
            'AC': 2,
            'BA': 3,
            'BB': 4,
            'BC': 5,
            'CA': 6,
            'CB': 7,
            'CC': 8}

  table2 = { '0' :'AA',
             '1' :'AB',
             '2' :'AC',
             '3' :'BA',
             '4' :'BB',
             '5' :'BC',
             '6' :'CA',
             '7' :'CB',
             '8' :'CC'}

  text = plaintext.split(sep="~")

  # initialize cipher
  cipher = ""

  # Iterate over index
  for p in range(0, len(text[0]),2):
    s = plaintext[p]+plaintext[p+1]

    value = table1[s]
    #print(value)
    value =  key[value]
    #print(value)
    value = table2[value]
    #print(value)
    cipher = cipher + value

  cipher = cipher + '~'+text[1]

  #return ciper
  return cipher
```

Output after Ecryption:

```
Ciphertext after Encryption :

BABBBCCACBCCAAABAC~1261564828
BABBBCAAABACCACBCC~2567393746
AAABACCACBCCBABBBC~4259082963
ACABAABABBBCCACCCB~2454973805
CCCBCABABBBCAAACAB~614313706
```

- **Step5: And decrypt the 5 ciphertext with the same key.**

With the help of the **decryptMessage(ciphertext, key) function** we decrypt the ciphertext by the same key which is used for encryption.

```
54 def decrptMessage(ciphertext, key):
55
56   table1 = {'AA': 0,
57             'AB': 1,
58             'AC': 2,
59             'BA': 3,
60             'BB': 4,
61             'BC': 5,
62             'CA': 6,
63             'CB': 7,
64             'CC': 8}
65
66   table2 = { '0' :'AA',
67              '1' :'AB',
68              '2' :'AC',
69              '3' :'BA',
70              '4' :'BB',
71              '5' :'BC',
72              '6' :'CA',
73              '7' :'CB',
74              '8' :'CC'}
75
76   text = ciphertext.split(sep="~")
77
78   # initialize cipher
79   plaintext = ""
80
81   # Iterate over index
82   for p in range(0, len(text[0]),2):
83     s = ciphertext[p] + ciphertext[p+1]
84
85     value = table1[s]
86
87     value = str(value)
88     for i in range(0, len(key)):
89       if(key[i] == value):
90         value = i
91         break
92
93     value = str(value)
94     value = table2[value]
95     plaintext = plaintext+value
96
97   plaintext = plaintext + '~' + text[1]
98   return plaintext
```

Output after Decryption:

```
Plaintext after Decryption :

AAABACBABBBCCACBCC~1261564828
AAABACCACBCCBABBBC~2567393746
CACBCCBABBBCAAABAC~4259082963
CCCBCAAAABACBABCBB~2454973805
BCBBBAAAABACCACCCB~614313706
```

- **Step6: And then finally do BruteForce attack on the same 5 ciphertext after generating the 9! Keys.**

For BruteForce attack, generate all the permutations of all keys and store them into a list. For every cipher we convert the plain text by the all permutation of the key and check and check if the plaintext is recognizable or not. If it is recognizable then check for other cipher text otherwise check for other key.

```
def Brute_force_attack(list_of_ciphertext):
  print("Do brute Force")

  key = "012345678"
  permutations_list = allPermutations(key)

  for cipher in list_of_ciphertext:
    for current_key in permutations_list:
      decrypt_plaintext = decrptMessage(cipher, current_key)
      valid = checkValidity(decrypt_plaintext)

      if(valid == 1 ):
        print("Cipher: "+ cipher + "  Key: " + current_key)
        break
```

```
Decrypt on 9! different keys to find list_of_plaintext given list_of_cipher
Do brute Force
Cipher: BABBBCCACBCCAAABAC~1261564828  Key: 345678012
Cipher: BABBBCAAABACCACBCC~2567393746  Key: 345678012
Cipher: AAABACCACBCCBABBBC~4259082963  Key: 345678012
Cipher: ACABAABABBBCCACCCB~2454973805  Key: 345678012
Cipher: CCCBCABABBBCAAACAB~614313706  Key: 345678012
```

Thank You…….!