NIST COLLEGE

Banepa, Kavre

Bachelor In Information Management

Seventh Semester

A
Lab Manual
on
Computer Security and Cyber Law

Compiled by

Er. Dinesh Ghemosu

1 LAB-1

FAMILIARIZATION WITH ENCRPYTION AND DECRYPTION

Objectives:

1. To implement ceaser cipher

1.1 Program 1

Program 1: WAP to implement Ceaser Cipher.

```
alpha ="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
print("Lenght of alpha: {}".format(len(alpha)))

# input in capitalize
str_in = input("Enter a word, like HELLO:")
print("str_in = ", str_in)

msg_cipher = " "
n = len(str_in)
print("n =", n)

for i in range(n):
    c = str_in[i]
    loc = alpha.find(c)
    print(i , c, loc)
    newloc = loc+3
    cipher_letter = alpha[newloc]
    msg_cipher += alpha[newloc]
    print(newloc, cipher_letter,msg_cipher)

print("Cipher:", msg_cipher)
```

Listing 1: Ceaser Cipher.

1.2 Program 2

Program 2: WAP to implement Ceaser Cipher with shift value.

```
alpha ="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
print("Lenght of alpha: {}".format(len(alpha)))

# input a word and it will change to uppercase
str_in = input("Enter a word, like HELLO:").upper()

# input a number in integer
shift = int(input("Input Shift value, like 3: "))

n = len(str_in)
msg_cipher = " "

for i in range(n):
    c = str_in[i]
    loc = alpha.find(c)
    print(i , c, loc)
    newloc = loc+shift  #can use newloc = (loc+shift) %26 and omit if
        condition below,will provide same result
```

```
    if newloc >=26:
        newloc -=26
    cipher_letter = alpha[newloc]
    msg_cipher += alpha[newloc]
    print(newloc, cipher_letter,msg_cipher)

print("cipher of palintext: ", msg_cipher)
```

Listing 2: Ceaser Cipher with shift value.

ROT13

ROT13 is nothing more than a Caesar cipher with a shift equal to 13 characters. In the script that follows, we will hardcode the shift to be 13 . If you run one cycle of ROT13, it changes HELLO to URYYB , and if you encrypt it again with the same process, putting in that URYYB , it'll turn back into HELLO , because the first shift is just by 13 characters and shifting by another 13 characters takes the total shift to 26 , which wraps right around, and that is what makes this one useful and important:

1.3 Program 3

Program 3: WAP to implement ROT13.

```
alpha ="ABCDEFGHIJKLMNOPQRSTUVWXYZ"

# input a word and it will change to uppercase
str_in = input("Enter a word, like HELLO:").upper()
shift = 13 #ROT13 has shift = 13

n = len(str_in)
msg_cipher = " "

for i in range(n):
    c = str_in[i]
    loc = alpha.find(c)
    newloc = (loc+shift)%26
    msg_cipher += alpha[newloc]

print("Obfuscasted version of plainword: ", msg_cipher)
```

Listing 3: ROT13.

1.4 Program 4

Program 4: Decrypting Ceaser Cipher.

```
alpha ="ABCDEFGHIJKLMNOPQRSTUVWXYZ"

# input a word and it will change to uppercase
str_in = input("Enter a word, like HELLO:").upper() #khoor give hello plaintext

for shift in range(26):
    n = len(str_in)
    msg_cipher = " "

    for i in range(n):
```

```
        c = str_in[i]
        loc = alpha.find(c)
        newloc = (loc + shift) % 26
        msg_cipher += alpha[newloc]
    print(shift, msg_cipher)s
```

Listing 4: Decrypting ceaser cipher.

2 LAB-2: Hashing

2.1 Program 5

Program 5: WAP to create MD5 hash.

```

import hashlib

md5hasher = hashlib.md5()
hash5 = md5hasher.hexdigest()
print("MD5 hash is: ", hash5)
```

Listing 5: MD5 hash.

2.2 Program 6

Program : WAP to create MD5 hash for bob and alice.

```

import hashlib

md5hasher = hashlib.md5(b'alice')
alice_hash = md5hasher.hexdigest()
print("Hash of text, alice is: \n", alice_hash)

md5hasher = hashlib.md5(b'bob')
bob_hash = md5hasher.hexdigest()
print("Hash of text, alice is: \n", bob_hash)
```

Listing 6: MD5 hash of Bob and Alice.

2.3 Program 7

Program 7: WAP to create MD5 hash for different word/text.

```
import hashlib

print("Hash of letter a is:\n",hashlib.md5(b'a').hexdigest()}
print("Hash of letter aa is:\n",hashlib.md5(b'aa').hexdigest()}
print("Hash of letter aaaaa is:\n",hashlib.md5(b'aaaaa').hexdigest()}
print("Hash of letter 100000 time of a is:\n",hashlib.md5(b'a'*100000).
    hexdigest()}
```

Listing 7: MD5 of different words.

2.4 Program 8

Program 8: WAP to create SHA-1 and SHA-256 hash .

```
import hashlib
print("SHA-1 has of word alice is:\n",hashlib.sha1(b'alice').hexdigest
    ()}
print("SHA-256 has of word alice is:\n",hashlib.sha1(b'alice').
    hexdigest()}
```

Listing 8: SHA-1 and SHA-256 Hash.

3 Lab-3: RSA

3.1 Program 9

Program 9: WAP to implement RSA .

```
# Python for RSA asymmetric cryptographic algorithm.
# For demonstration, values are
# relatively small compared to practical application
import math


def gcd(a, h):
  temp = 0
  while(1):
    temp = a % h
    if (temp == 0):
      return h
    a = h
    h = temp


p = 3
q = 7
n = p*q
e = 2
phi = (p-1)*(q-1)

while (e < phi):

  # e must be co-prime to phi and
  # smaller than phi.
  if(gcd(e, phi) == 1):
    break
  else:
    e = e+1

# Private key (d stands for decrypt)
# choosing d such that it satisfies
# d*e = 1 + k * totient

k = 2
d = (1 + (k*phi))/e

# Message to be encrypted
```

```
msg = 12.0

print("Message data = ", msg)

# Encryption c = (msg ^ e) % n
c = pow(msg, e)
c = math.fmod(c, n)
print("Encrypted data = ", c)

# Decryption m = (c ^ d) % n
m = pow(c, d)
m = math.fmod(m, n)
print("Original Message Sent = ", m)
```

Listing 9: RSA.

4 LAB-4: Transpositional Cipher

4.1 Program 10

Program 10: WAP to implement Transpositional Cipher.

```
# Implementation of Columnar Transposition
import math

key = "HACK"

# Encryption
def encryptMessage(msg):
  cipher = ""

  # track key indices
  k_indx = 0

  msg_len = float(len(msg))
  msg_lst = list(msg)
  key_lst = sorted(list(key))

  # calculate column of the matrix
  col = len(key)

  # calculate maximum row of the matrix
  row = int(math.ceil(msg_len / col))

  # add the padding character '_' in empty
  # the empty cell of the matix
  fill_null = int((row * col) - msg_len)
  msg_lst.extend('_' * fill_null)

  # create Matrix and insert message and
  # padding characters row-wise
  matrix = [msg_lst[i: i + col]
      for i in range(0, len(msg_lst), col)]

  # read matrix column-wise using key
  for _ in range(col):
    curr_idx = key.index(key_lst[k_indx])
```

```
    cipher += ''.join([row[curr_idx]
            for row in matrix])
    k_indx += 1

  return cipher

# Decryption
def decryptMessage(cipher):
  msg = ""

  # track key indices
  k_indx = 0

  # track msg indices
  msg_indx = 0
  msg_len = float(len(cipher))
  msg_lst = list(cipher)

  # calculate column of the matrix
  col = len(key)

  # calculate maximum row of the matrix
  row = int(math.ceil(msg_len / col))

  # convert key into list and sort
  # alphabetically so we can access
  # each character by its alphabetical position.
  key_lst = sorted(list(key))

  # create an empty matrix to
  # store deciphered message
  dec_cipher = []
  for _ in range(row):
    dec_cipher += [[None] * col]

  # Arrange the matrix column wise according
  # to permutation order by adding into new matrix
  for _ in range(col):
    curr_idx = key.index(key_lst[k_indx])

    for j in range(row):
      dec_cipher[j][curr_idx] = msg_lst[msg_indx]
      msg_indx += 1
    k_indx += 1

  # convert decrypted msg matrix into a string
  try:
    msg = ''.join(sum(dec_cipher, []))
  except TypeError:
    raise TypeError("This program cannot",
            "handle repeating words.")

  null_count = msg.count('_')

  if null_count > 0:
    return msg[: -null_count]

  return msg
```

```python

# Driver Code
msg = "Attack is Tonight!"

cipher = encryptMessage(msg)

print("Plain text message is: {}".format(msg))
print("Encrypted Message: {}".format(cipher))
print("Decryped Message: {}".format(decryptMessage(cipher)))
```

Listing 10: Transpositional Cipher.