Roshan Mundekar

MSC CS PART 1

ROLL NO - 508

mundekarroshan566@gmail.com

SUBJECT - CRYPTOGRAPHY AND CRYPTANALYSIS

Ramniranjan Jhunjhunwala College
AUTONOMOUS
(UNIVERSITY OF MUMBAI)
MUMBAI - 400019

DEPARTMENT OF COMPUTER SCIENCE

CERTIFICATE

Exam Seat No. **A-231**

CERTIFIED that the practicals, assignment duly signed, were performed by Ms. **Roshan J. Mundekar** Roll No **508** of M.Sc Part I class in the Computer Science Laboratory of G. N. Khalsa College, Mumbai during academic year 2023-2024.
She has completed the course of Laboratory assignments in Computer Science as contained in the course prescribed by the University of Mumbai

Sign. Of the Student

Date_____

Professor-in-charge
1)_____
Date_____
2)_____
Date_____

Head of Dept.
Computer Science

Date_____

Sign of Examiner's
1)_____
Date_____
2)_____
Date_____

INDEX

PRACTICAL 1

AIM

Program to implement password salting and hashing to create secure passwords.

THEORY

Salting

Password salting is a technique to protect passwords stored in databases by adding a string of 32 or more characters and then hashing them. Salting prevents hackers who breach an enterprise environment from reverse-engineering passwords and stealing them from the database.

How password salting works?

With password salting, a random piece of data is added to the password before it runs through the hashing algorithm, making it unique and harder to crack.When using both hashing and salting, even if two users choose the same password, salting adds random characters to each password when the users enter them. As a result, completely different hashes are generated to prevent the passwords and accounts from being compromised.To prevent password attacks, salts must be unique and random for each login. Encrypting password storage at rest can provide additional defense in depth, even if a hacker were able to recalculate hashes of common password lists using a given salt for a password.

Which attacks can password salting prevent or minimize?

Attackers use many tools to crack password hashes. This is because hash tables are designed to be fast but are not necessarily secure. By adding randomness to the original plaintext password value before hashing, salting ensures that a different hashed value is generated.

Salting is an additional layer of security to prevent, or at least minimize, the possibility of password compromise by the following three primary attack vectors.

1. Brute-force attacks
   In this type of attack, hackers try to guess every possible password combination and then run these combinations through a hashing algorithm. Once a match is found, they can find the original password. Since password salting creates unique hashes for each password, the attacker cannot guess the original password by brute force.

2. Dictionary attacks
   Dictionary attacks are the advanced version of brute-force attacks. In this type of attack, bad actors try the most common password word and character combinations. They use a prearranged word list with their computed hash and then compare the hashes from a stolen password table with every hash on the list. If they manage to find a match, they can easily find the password. Salting makes this process more difficult and mitigates dictionary attacks.

3. Rainbow table attacks
   Attackers often use rainbow tables to crack unsalted hashes. A rainbow table is a pre-computed database of decrypted hash passwords, which attackers can search to find the desired hash. But, when salting is used for each password, attackers will fail. Even if they know the salt, they would still need to build a rainbow table for each salt. This is how password salting helps prevent rainbow table attacks.

Hashing

Hashing is the transformation of a string of characters into a usually shorter fixed-length value or key that represents the original string. Hashing is used to index and retrieve items in a database because it is faster to find the item using the shorter hashed key than to find it using the original value

Hashing is a technique or process of mapping keys, and values into the hash table by using a hash function. It is done for faster access to elements. The efficiency of mapping depends on the efficiency of the hash function used.

Let a hash function H(x) maps the value

at the index x%10 in an Array. For example if the list of values is [11,12,13,14,15] it will be stored at positions {1,2,3,4,5} in the array or Hash table respectively.

CODE

```
import bcrypt

pwd = input('Enter the password: ')
falsePwd = "falsepassword"
bytePwd = pwd.encode('utf-8')
byteFalsepwd = falsePwd.encode('utf-8')

# Generate salt
mySalt = bcrypt.gensalt()

# Hash password
hash = bcrypt.hashpw(bytePwd, mySalt)
print('Hashed password : ', hash)
print('Matching hashed password with entered password : ' ,
bcrypt.checkpw(bytePwd, hash))
print('Matching hashed password with false password : ' ,
bcrypt.checkpw(byteFalsepwd, hash))
```

OUTPUT

```
========== RESTART: G:\Gursheen\221\cryptography\P1\
salting&hashing.py =========
Enter the password: hello
Hashed password :  b'$2b$12$eMWukCQp0nHpGuAjRoK6p.Ewv
rQ3Hhrq0TMBwOn5XU/TBErFOPcSW'
Matching hashed password with entered password :  True
Matching hashed password with false password :  False
>>>
```

PRACTICAL 2

AIM

Program to implement following classical ciphers -

a. Caesar Cipher
b. Vigenère Cipher
c. Affine Cipher

THEORY

Substitution cipher

Hiding some data is known as encryption. When plain text is encrypted it becomes unreadable and is known as ciphertext. In a Substitution cipher, any character of plain text from the given fixed set of characters is substituted by some other character from the same set depending on a key. For example with a shift of 1, A would be replaced by B, B would become C, and so on.
Note: Special case of Substitution cipher is known as Caesar cipher where the key is taken as 3.

Mathematical representation

The encryption can be represented using modular arithmetic by first transforming the letters into numbers, according to the scheme, A = 0, B = 1,…, Z = 25.
Encryption of a letter by a shift n can be described mathematically as.

(Encryption)

$$E_n(x) = (x + n) mod\ 26$$

(Decryption)

$$D_n(x) = (x - n) mod\ 26$$

Algorithm for Substitution Cipher:

Input:

- A String of both lower and upper case letters, called PlainText.
- An Integer denoting the required key.

Procedure:

- Create a list of all the characters.
- Create a dictionary to store the substitution for all characters.
- For each character, transform the given character as per the rule, depending on whether we're encrypting or decrypting the text.
- Print the new string generated.

CODE

```
#SHIFT CIPHER
def encrypt_words(plain_text,key):
	cipher_text=''
	for word in plain_text:
		for i in word:
			if i.isupper():
				val=ord(i)-65
				enc_word=chr(65+(val+key)%26)
			else:
				val=ord(i)-97
				enc_word=chr(97+(val+key)%26)
		cipher_text+=enc_word
	print('Encrypted Text : ',cipher_text)
	return cipher_text

def decrypt_words(cipher_text,key):
	plain_text=''
	for word in cipher_text:
		for i in word:
			if i.isupper():
				val=ord(i)-65
				enc_word=chr(65+(val-key)%26)
			else:
				val=ord(i)-97
				enc_word=chr(97+(val-key)%26)
		plain_text+=enc_word
	print('Decrypted Text : ',plain_text)
plain_text=input('Enter the plain text to be encrypted:').split()
key=int(input('Enter the key for shift cipher:'))
cipher_text=encrypt_words(plain_text,key)
decrypt_words(cipher_text,key)
```

OUTPUT

```
========== RESTART: G:\Gursheen\221\cryptography\P2\shiftcipher.py
Enter the plain text to be encrypted:gursheen
Enter the key for shift cipher:4
Encrypted Text :  kyvwliir
Decrypted Text :  gursheen
>>>
```

THEORY

Vignere cipher

- Vigenere Cipher is a method of encrypting alphabetic text. It uses a simple form of polyalphabetic substitution. A polyalphabetic cipher is any cipher based on substitution, using multiple substitution alphabets. The encryption of the original text is done using the Vigenère square or Vigenère table.
- The table consists of the alphabets written out 26 times in different rows, each alphabet shifted cyclically to the left compared to the previous alphabet, corresponding to the 26 possible Caesar Ciphers.
- At different points in the encryption process, the cipher uses a different alphabet from one of the rows.
- The alphabet used at each point depends on a repeating keyword.

Encryption

The first letter of the plaintext, G is paired with A, the first letter of the key. So use row G and column A of the Vigenère square, namely G. Similarly, for the second letter of the plaintext, the second letter of the key is used, the letter at row E, and column Y is C. The rest of the plaintext is enciphered in a similar fashion.

Decryption

Decryption is performed by going to the row in the table corresponding to the key, finding the position of the ciphertext letter in this row, and then using the column's label as the plaintext. For example, in row A (from AYUSH), the ciphertext G appears in column G, which is the first plaintext letter. Next, we go to row Y (from AYUSH), locate the ciphertext C which is found in column E, thus E is the second plaintext letter.

Algorithm

A more easy implementation could be to visualize Vigenère algebraically by converting [A-Z] into numbers [0–25].

- Encryption

The plaintext(P) and key(K) are added modulo 26.
$E_i = (P_i + K_i) \bmod 26$

- Decryption
  $D_i = (E_i - K_i + 26) \bmod 26$

CODE

```
#VIGNERE CIPHER
import math

def encrypt_words(plain_text,key):
        cipher_text=''
        n=len(plain_text)
        ceil_val=math.ceil(n/len(key))
        key=ceil_val*key
        for i in range(n):
                if plain_text[i].islower():
                        pi=ord(plain_text[i])-97
                        ki=ord(key[i])-97
                        ei=(pi+ki)%26
                        cipher_text+=chr(ei+97)
                else:
                        pi=ord(plain_text[i])-65
                        ki=ord(key[i])-65
                        ei=(pi+ki)%26
                        cipher_text+=chr(ei+65)
        print('Encrypted Text : ',cipher_text)
        return cipher_text

def decrypt_words(cipher_text,key):
        plain_text=''
        n=len(cipher_text)
        ceil_val=math.ceil(n/len(key))
        key=ceil_val*key
        for i in range(n):
                if cipher_text[i].islower():
                        ei=ord(cipher_text[i])-97
                        ki=ord(key[i])-97
```

```
                di=(ei-ki+26)%26
                plain_text+=chr(di+97)
        else:
                ei=ord(cipher_text[i])-65
                ki=ord(key[i])-65
                di=(ei-ki+26)%26
                plain_text+=chr(di+65)
    print('Decrypted  Text  : ',plain_text)

plain_text=input('Enter the plain text to encrypt:')
key=input('Enter the key for encryption:')
enc=encrypt_words(plain_text,key)
decrypt_words(enc,key)
```

OUTPUT

```
========== RESTART: G:\Gursheen\221\cryptography\P2\vignere
cipher.py ==========
Enter the plain text to encrypt:gursheen
Enter the key for encryption:vignerecipher
Encrypted Text :  bcxflvip
Decrypted  Text  :  gursheen
>>>
```

THEORY

Affine cipher

- The Affine cipher is a type of monoalphabetic substitution cipher, wherein each letter in an alphabet is mapped to its numeric equivalent, encrypted using a simple mathematical function, and converted back to a letter.
- The formula used means that each letter encrypts to one other letter, and back again, meaning the cipher is essentially a standard substitution cipher with a rule governing which letter goes to which.
- The whole process relies on working modulo m (the length of the alphabet used). In the affine cipher, the letters of an alphabet of size m are first mapped to the integers in the range 0 … m-1.
- The 'key' for the Affine cipher consists of 2 numbers, we'll call them a and b. The following discussion assumes the use of a 26 character alphabet (m = 26). a should be chosen to be relatively prime to m (i.e. a should have no factors in common with m).

Encryption

It uses modular arithmetic to transform the integer that each plaintext letter corresponds to into another integer that correspond to a ciphertext letter. The encryption function for a single letter is

E (x) = (a x + b) mod m

modulus m: size of the alphabet

a and b: key of the cipher.

a must be chosen such that a and m are coprime.

Decryption

In deciphering the ciphertext, we must perform the opposite (or inverse) functions on the ciphertext to retrieve the plaintext. Once again, the first step is to convert each of the ciphertext letters into their integer values. The decryption function is

D (x) = a^-1 (x - b) mod m

a^-1 : modular multiplicative inverse of a modulo m. i.e., it satisfies the equation 1 = a a^-1 mod m .

To find a multiplicative inverse
We need to find a number x such that:
If we find the number x such that the equation is true, then x is the inverse of a, and we call it a^-1. The easiest way to solve this equation is to search each of the numbers 1 to 25, and see which one satisfies the equation.
[g,x,d] = gcd(a,m); % we can ignore g and d, we dont need them
x = mod(x,m);
If you now multiply x and a and reduce the result (mod 26), you will get the answer 1. Remember, this is just the definition of an inverse i.e. if a*x = 1 (mod 26), then x is an inverse of a (and a is an inverse of x)

CODE

```
#AFFINE CIPHER
def encrypt_words(plain_text,a,b):
	cipher_text=''
	for i in plain_text:
		if i.isupper():
			x=ord(i)-65
			enc_val=(a*x+b)%26
			cipher_text+=chr(enc_val+65)
		else:
			x=ord(i)-97
			enc_val=(a*x+b)%26
			cipher_text+=chr(enc_val+97)
	print('Encrypted Text : ',cipher_text)
	return cipher_text

def decrypt_words(cipher_text,a,b):
	c=0
	for i in range(1,27):
```

```
            if (a*i)%26==1:
                c=i
    plain_text=''
    for i in cipher_text:
        if i.isupper():
            y=ord(i)-65
            enc_val=(c*(y-b))%26
            plain_text+=chr(enc_val+65)
        else:
            y=ord(i)-97
            enc_val=(c*(y-b))%26
            plain_text+=chr(enc_val+97)
    print('Decrypted Text : ',plain_text)

plain_text=input('Enter the plain text to be encrypted:')
a=int(input('Enter the value of a:'))
b=int(input('Enter the value of b:'))
enc=encrypt_words(plain_text,a,b)
decrypt_words(enc,a,b)
```

OUTPUT

```
========== RESTART: G:\Gursheen\221\cryptography\P2\
affinecipher.py ==========
Enter the plain text to be encrypted:AffineCipher
Enter the value of a:3
Enter the value of b:5
Encrypted Text :  FuudsrLdyare
Decrypted Text :  AffineCipher
>>>
```

PRACTICAL 3

AIM

Program to demonstrate cryptanalysis of Shift Cipher.

THEORY

Cryptanalysis

Cryptanalysis refers to the process of analyzing information systems in order to understand hidden aspects of the systems. Cryptanalysis is used to breach cryptographic security systems and gain access to the contents of encrypted messages, even if the cryptographic key is unknown.

Cryptanalytic attacks

The attacks rely on nature of the algorithm and also knowledge of the general characteristics of the plaintext, i.e., plaintext can be a regular document written in English or it can be a code written in Java. Therefore, nature of the plaintext should be known before trying to use the attacks.

The Five Types of Cryptanalytic Attacks -

- Known-Plaintext Analysis (KPA) :
  In this type of attack, some plaintext-ciphertext pairs are already known. Attacker maps them in order to find the encryption key. This attack is easier to use as a lot of information is already available.
- Chosen-Plaintext Analysis (CPA) :
  In this type of attack, the attacker chooses random plaintexts and obtains the corresponding ciphertexts and tries to find the encryption key. Its very simple to implement like KPA but the success rate is quite low.

- Ciphertext-Only Analysis (COA) :

In this type of attack, only some cipher-text is known and the attacker tries to find the corresponding encryption key and plaintext. Its the hardest to implement but is the most probable attack as only ciphertext is required.

- Man-In-The-Middle (MITM) attack :
  In this type of attack, attacker intercepts the message/key between two communicating parties through a secured channel.
- Adaptive Chosen-Plaintext Analysis (ACPA) :
  This attack is similar CPA. Here, the attacker requests the cipher texts of additional plaintexts after they have ciphertexts for some texts.

CODE - SHIFT CIPHER CRYPTANALYSIS

```
def cryptanalysis():
    cipher_text=input('Enter the cipher text for cryptanalysis : ')
    for k in range(26):
    plain_text=''
    for letter in cipher_text:
    if letter==' ':
        plain_text+=letter
    else:
        c=ord(letter)-65
        e=(c-k)%26
        plain_text+=chr(e+65)
    print('With key = ',k,plain_text)
cryptanalysis()
```

OUTPUT

```
======== RESTART: G:\Gursheen\221\cryptography\P3\caesarcryptanalysis.py ==
Enter the cipher text for cryptanalysis : GUVF VF ZL FRPERG ZRFFNTR
With key =  0 GUVF VF ZL FRPERG ZRFFNTR
With key =  1 FTUE UE YK EQODQF YQEEMSQ
With key =  2 ESTD TD XJ DPNCPE XPDDLRP
With key =  3 DRSC SC WI COMBOD WOCCKQO
With key =  4 CQRB RB VH BNLANC VNBBJPN
With key =  5 BPQA QA UG AMKZMB UMAAIOM
With key =  6 AOPZ PZ TF ZLJYLA TLZZHNL
With key =  7 ZNOY OY SE YKIXKZ SKYYGMK
With key =  8 YMNX NX RD XJHWJY RJXXFLJ
With key =  9 XLMW MW QC WIGVIX QIWWEKI
With key =  10 WKLV LV PB VHFUHW PHVVDJH
With key =  11 VJKU KU OA UGETGV OGUUCIG
With key =  12 UIJT JT NZ TFDSFU NFTTBHF
With key =  13 THIS IS MY SECRET MESSAGE
With key =  14 SGHR HR LX RDBQDS LDRRZFD
With key =  15 RFGQ GQ KW QCAPCR KCQQYEC
With key =  16 QEFP FP JV PBZOBQ JBPPXDB
With key =  17 PDEO EO IU OAYNAP IAOOWCA
With key =  18 OCDN DN HT NZXMZO HZNNVBZ
With key =  19 NBCM CM GS MYWLYN GYMMUAY
With key =  20 MABL BL FR LXVKXM FXLLTZX
With key =  21 LZAK AK EQ KWUJWL EWKKSYW
With key =  22 KYZJ ZJ DP JVTIVK DVJJRXV
With key =  23 JXYI YI CO IUSHUJ CUIIQWU
With key =  24 IWXH XH BN HTRGTI BTHHPVT
With key =  25 HVWG WG AM GSQFSH ASGGOUS
```

PRACTICAL 4

AIM

Write a program to implement the AES algorithm for File Encryption and Decryption.

THEORY

AES File encryption

The AES Encryption algorithm (also known as the Rijndael algorithm) is a symmetric block cipher algorithm with a block/chunk size of 128 bits. It converts these individual blocks using keys of 128, 192, and 256 bits. Once it encrypts these blocks, it joins them together to form the ciphertext.

It is based on a substitution-permutation network, also known as an SP network. It consists of a series of linked operations, including replacing inputs with specific outputs (substitutions) and others involving bit shuffling (permutations).

What are the Features of AES?

1. SP Network: It works on an SP network structure rather than a Feistel cipher structure, as seen in the case of the DES algorithm.
2. Key Expansion: It takes a single key up during the first stage, which is later expanded to multiple keys used in individual rounds.
3. Byte Data: The AES encryption algorithm does operations on byte data instead of bit data. So it treats the 128-bit block size as 16 bytes during the encryption procedure.
4. Key Length: The number of rounds to be carried out depends on the length of the key being used to encrypt data. The 128-bit key size has ten rounds, the 192-bit key size has 12 rounds, and the 256-bit key size has 14 rounds.

CODE

```
from Crypto.Cipher import AES
import binascii, os
text_file = open("demo_textfile.txt", "r")
text = text_file.read()
print("File contents -",'\n'+text)
msg = bytes(text, encoding='utf-8')

#secret key
secretKey = os.urandom(32)

#Encryption
aesCipher = AES.new(secretKey, AES.MODE_GCM)
ciphertext, authTag = aesCipher.encrypt_and_digest(msg)
encryptedMsg = ciphertext, aesCipher.nonce, authTag

#Decryption
(ciphertext, nonce, authTag) = encryptedMsg
aesCipher = AES.new(secretKey, AES.MODE_GCM, nonce)
plaintext = aesCipher.decrypt_and_verify(ciphertext, authTag)
ct = binascii.hexlify(encryptedMsg[0]).decode('utf-8')

pt = plaintext.decode('utf-8')
#Print data in new file
file_text = open("Decryption_file.txt","w")
text2 = file_text.write("Encrypted Data:-\n")
text2 = file_text.write(str(encryptedMsg))
text2 = file_text.write("\n\nDecryption or Plaintext:-\n")
text2 = file_text.write(pt)
file_text.close()
```

OUTPUT

```
========== RESTART: G:/Gursheen/221/cryptography/P4/aes.py ==========
File contents -
The Advanced Encryption Standard (AES) is a symmetric block cipher chosen by the U.S. government to protect classified information.

The National Institute of Standards and Technology (NIST) started development of AES in 1997 when it announced the need for an alternative to the Data Encryption Standard (DES), which was starting to become vulnerable to brute-force attacks.

NIST stated that the newer, advanced encryption algorithm would be unclassified and must be "capable of protecting sensitive government information well into the [21st] century." It was intended to be easy to implement in hardware and software, as well as in restricted environments -- such as a smart card -- and offer decent defenses against various attack techniques.
>>>
```

Decryption_file - Notepad

File Edit Format View Help

```
Encrypted Data:-
(b'M\x8c\x0f~+\xfb)\x13\xe5\xef&X!2\xc0\x17\xaf\x8f\xb82C\xb503\xd9\xbc&
\xa0\xa3*EXie}\xeb\xc8d\xfb\xea\x1b2\x88\xcf\x81`\x97d<4\xa4*7\xdcS\x1eT\xce\x83w
\x04\xb1\x8b^\xdf\xd2\xd3\xda\n\xda\x81\xb1\x8eD\x96\xcc+\xceT\xc6\xe3<\x8c\xf7\x18\xfe
\x84\x83_7\'\xf54\x98\x02\xcc\x8c\x8c=\xecm\xe0\x1dq\x03\xafb\xa6E\x90\x18\xf4q\x0e
\xf8\x13&\r\x15\xc6\x15\x8c\xbc\x06{\xce\xa8\xc7c\xf1\xb4^"\t\x086\xec\xcdj\x1c\xb9o\xab
\xfc\xe8\xd9\x80\x11\xeb\xc8\x05$\xcb\xa9\xa0<\xfd\xe6\xf6\xd38\xdac\xf4\xb9\xd1\x84\xfa
\x868\xd9\xcfVP@q\xd8(\xaf\xad\xdb[\x8d\x99\x84\xf6\x07_\\"\xd9\x10\xf8)\xcaj\x8f
\xd4\xd3\xa8\x8c\xdf2c\x0ehd\xb8LI\x10j\xe0\xb7\x1fR\xeb\x88\x96X\xbe\x00\x18
\xe9\x17\xe4%\xfa\xce\xcfL\xc4\x02UZ\x9eI\xf6\xd2\x8f\xfb\x99\xf2\x93\xfc
\xa9\x80\x87\xc4\x17\xa4\xe7]J\xad\xfeHL\xf2/\x81\x12g\xd6\xba\x0e\xcc\xa2\xd8\xdcga.\xac
\'\xb8\x85\xafu-!\xd6A\x8a\xd77\xe8M\xbc\x8b:x@\x1f\xd7\xf8\xe1\xd9\x0e\xabqqOX\xdb
\xf8+nz\x8aJ\xb5P\xd6\x82\xc1/\xae\xdb\xb9T\xf6\x96\x95\xd7y\xbe
\xa4\xe4\xc2\xb22\xa0\x05\xd8"^q\xb72e}\xbb8\xbb\xb6\xeaDe_\x05\x98/e\x9f>\x997m\x90\x8a
\xc8\xf2t\x1cI\xd6>\xe0\xadI\xec\xac\x18N\x8bk\x17\xbd\xf2\nR\x01p-"\x1f\xd7\xab~\x8e
\xd4\xaa\x8e+c\xf8\xf9s\x80}\xcb\x94\xa5\x9b*Hag/\xcb[\xc1\xe1$;g\xb6.;\xec/x\xba
\xc2\x03\xce\x16\r\xc28\x13,\xc62 \x8b\xf1\xb6$B-<:\x1d\xb0-Gi\xcbRT\x11n\xff\xdf\xeb
\x06N1\xe8\x99c\xc4\xc9\xeefa\x1c\x1cln\n\xd4N\x1f\xf8\xe4\xc2\xbf[\x8c\x9e\xbc2\xfb\xfa
\xa8Xf\xa3\x8b\xe0\xd4,\x0fd\x9f\x92\xb31\xdcn\x1f\xe9\xc3\x08\x01\x03\xec\xb4\xd8\xfe
\xe6,\xb2g\xf9\xac\x95I&\xd8\xbeb~\xb8\x981\xad\xa2=\xdb\x801\x8c\x01\xe3\x8b_\xcc
\xe2\xae\xa9\'~\xb1\xc4F\x0fu\r\x9cD\xff\x04\x16i\xc5\x1f\x86\x0e\xe3JW\x94V\x90\xab
\x13\xb1Y\xf1\x81\xe7\xab\x08\\\x86\xd7u\n\xfd~@\xe8,\x0f\x0b\x03\x0c\xa7f\xa4s5"\xe0R
\xe8\xda\xf7\xcf\x16F\x03\xe1H\xf3\x01V\xf7AC\x1e$\x16\xc45\xa0\x86\xd7\x8bu
\x80\xa6\xb2\x98\xc9\xe2\x84]\xbcS\xf15\'\xa6\xae\x87\xd4oj\x0f\xbe/\xa10\xb7n\xbc\xca
\x98\xc1\xca\xb1T)\x08\x1b\xa3@\xdbc\xa4\xc0\x83\xe8\x8f_yxE\x16GP7\xdc\x83\t
\xb0\xc0\x05\xc91\xb3:\xa9\\G\x88\xc6\x1eV\x02\x1c\x0e>\x11\xb1X\xbcS\x1a\xd8\xef\x8fq
\xb3@!\xfe\x16%\xd1\x85@\xf312P\xbc\xa5\xf7`\x9e\x16rpMH;\xb6\xda\xac:c
\xd4\xc0\xb7\x12\xea\xa0\x0e[', b'\x03b}\x15Y\xf5\xab!p\xfc\xa3\xf9\xb7\x92\xb1\x82',
b'\x87\x97\xb6sfsYB\xc0G\x92}ui\xf8\x87')
```

Decryption or Plaintext:-
The Advanced Encryption Standard (AES) is a symmetric block cipher chosen by the U.S. government to protect classified information.

The National Institute of Standards and Technology (NIST) started development of AES in 1997 when it announced the need for an alternative to the Data Encryption Standard (DES), which was starting to become vulnerable to brute-force attacks.

NIST stated that the newer, advanced encryption algorithm would be unclassified and must be "capable of protecting sensitive government information well into the [21st] century." It was intended to be easy to implement in hardware and software, as well as in restricted environments -- such as a smart card -- and offer decent defenses against various attack techniques.

PRACTICAL 5

AIM

Program to implement Steganography for hiding messages inside the image file.

THEORY

Steganography

A steganography technique involves hiding sensitive information within an ordinary, non-secret file or message, so that it will not be detected. The sensitive information will then be extracted from the ordinary file or message at its destination, thus avoiding detection. Steganography is an additional step that can be used in conjunction with encryption in order to conceal or protect data.

Different Types of Steganography

1. Text Steganography − There is steganography in text files, which entails secretly storing information. In this method, the hidden data is encoded into the letter of each word.
2. Image Steganography − The second type of steganography is image steganography, which entails concealing data by using an image of a different object as a cover. Pixel intensities are the key to data concealment in image steganography.

Since the computer description of an image contains multiple bits, images are frequently used as a cover source in digital steganography.

CODE

```
from stegano import lsb

imgpath=input('Enter the image name(with file extension) : ')
msg=input('Enter message to be hidden : ')
secret=lsb.hide(imgpath,msg)
secret.save('image_with_hidden_text.png')
print('Image containing message saved.')
clear_message=lsb.reveal("image_with_hidden_text.png")
print('Verifying the message from saved image : ',clear_message)
```

OUTPUT

```
========== RESTART: G:\Gursheen\221\cryptography\P5\steganography.py ==
========
Enter the image name(with file extension) : steganography_img.png
Enter message to be hidden : THIS IMAGE HAS A SECRET MESSAGE
Image containing message saved.
Verifying the message from saved image :  THIS IMAGE HAS A SECRET MESSAGE
>>>
```

ORIGINAL IMAGE WITH SIZE

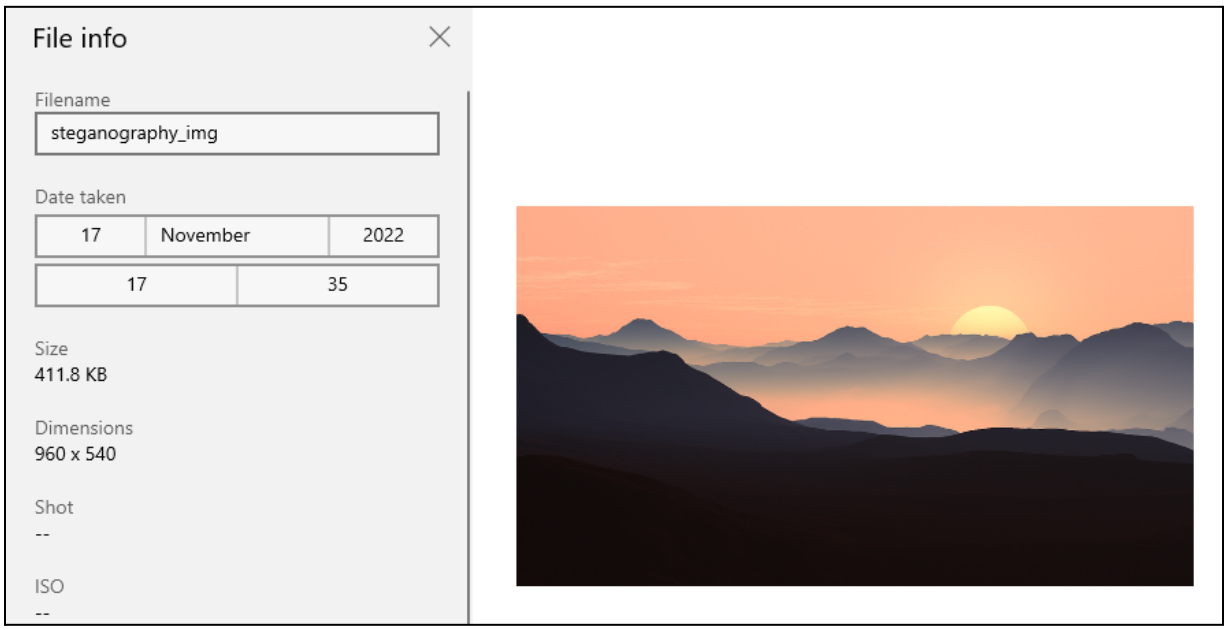
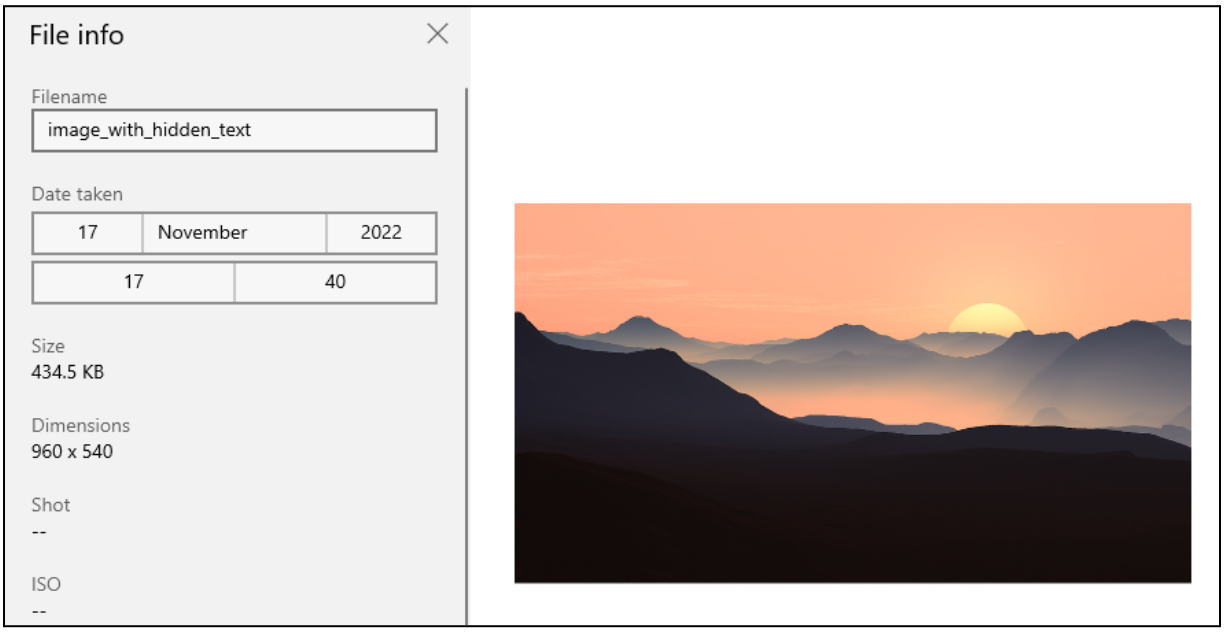


IMAGE GENERATED AFTER STEGANOGRAPHY WITH SIZE



PRACTICAL 6

AIM

Write a program to implement HMAC signatures.

THEORY

HMAC

Hash-based message authentication code (or HMAC) is a cryptographic authentication technique that uses a hash function and a secret key.
With HMAC, you can achieve authentication and verify that data is correct and authentic with shared secrets, as opposed to approaches that use signatures and asymmetric cryptography.

Applications:

- Verification of e-mail address during activation or creation of an account.
- Authentication of form data that is sent to the client browser and then submitted back.
- HMACs can be used for Internet of things (IoT) due to less cost.
- Whenever there is a need to reset the password, a link that can be used once is sent without adding a server state.
- It can take a message of any length and convert it into a fixed-length message digest. That is even if you got a long message, the message digest will be small and thus permits maximizing bandwidth.

CODE

```
import hmac
import hashlib
import secrets

#Initial sent message
sent_msg = input('Enter message:')
key = secrets.token_bytes(100)
s_md_1 =hmac.new(key=key,msg=sent_msg.encode(),digestmod=hashlib.md5)
init_msg_digest = s_md_1.hexdigest()

# Received Message
received = sent_msg
r_md_1 =hmac.new(key=key,msg=received.encode(),digestmod=hashlib.md5)
recv_msg_digest = r_md_1.hexdigest()

#Comparing sent and received messages
print("----Before Tampering----")
print('Is the message received without any
tampering:',hmac.compare_digest(init_msg_digest,recv_msg_digest))
#Tampered Message
tampered_msg = sent_msg[1:]
md_2 =
hmac.new(key=key,msg=tampered_msg.encode(),digestmod=hashlib.md5)
tampered_msg_digest = md_2.hexdigest()

#Comparing after tampering
print("----After Tampering----")
print('Is the message received without any
tampering:',hmac.compare_digest(init_msg_digest,tampered_msg_digest))
```

OUTPUT

```
================ RESTART: G:/Gursheen/221/crypto
graphy/P6/HMAC.py ================
>>>
Enter message : GURSHEEN
----------------Before Tampering----------------
Is the message received without any tampering :  True

----------------After Tampering----------------
Is the message received without any tampering :  False
>>>
```

PRACTICAL 7

AIM

Write a program to implement -

a. ElGamal Cryptosystem
b. Euclidean Algorithm

THEORY

ELGAMAL CRYPTOSYSTEM

ElGamal encryption is a public-key cryptosystem. It uses asymmetric key encryption for communicating between two parties and encrypting the message. This cryptosystem is based on the difficulty of finding discrete logarithms in a cyclic group that is even if we know ga and gk, it is extremely difficult to compute gak.

EUCLIDEAN ALGORITHM

The Euclidean algorithm is a way to find the greatest common divisor of two positive integers. The GCD of two numbers is the largest number that divides both of them. A simple way to find GCD is to factorize both numbers and multiply common prime factors.

CODE - ELGAMAL ALGORITHM

```
def generate_e2():
        e2=e1**d%p
        return e2

def generate_c1():
        c1=e1**r%p
        return c1

def generate_c2():
        c2=(pt*e2**r)%p
        return c2

def encryption():
        ct=(c1,c2)
        return ct

def decryption():
        temp=c1**d
        temp_inv=None
        for i in range(1,p):
        if(temp*i)%p==1:
        temp_inv=i
        break

        dpt=(c2*temp_inv)%p
        return dpt

p=int(input("Enter 1st part of public key : "))
e1=int(input("Enter 2nd part of public key : "))
d=int(input("Enter a private key : "))
r=int(input("Enter a random integer key :"))
```

```
pt=int(input("Enter the plain text : "))

e2=generate_e2()
c1=generate_c1()
c2=generate_c2()

print('Encrypted Text : ',encryption())
print('Decrypted Text : ',decryption())
```

OUTPUT

```
============== RESTART: G:/Gursheen/221/crypto
graphy/P7/elgamal.py ============
Enter 1st part of public key : 11
Enter 2nd part of public key : 2
Enter a private key : 3
Enter a random integer key :4
Enter the plain text : 7
Encrypted Text :  (5, 6)
Decrypted Text :  7
>>>
```

CODE - EUCLIDEAN ALGORITHM

```
def gcd(a,b):
        if a==0:
        return b
        else:
        return gcd(b%a,a)

a=int(input('Enter a :'))
b=int(input('Enter b:'))
print('GCD of a and b : ',gcd(a,b))
```

OUTPUT

```
============== RESTART: G:\Gursheen\221\
cryptography\euclidsalgo.py ==============
Enter a :662
Enter b:414
GCD of a and b :  2
>>>
```

PRACTICAL 8

AIM

Program to implement RSA encryption/decryption.

THEORY

RSA

RSA algorithm is asymmetric cryptography algorithm. Asymmetric actually means that it works on two different keys i.e. Public Key and Private Key. As the name describes that the Public Key is given to everyone and the Private key is kept private.

An example of asymmetric cryptography :

1. A client (for example browser) sends its public key to the server and requests for some data.
2. The server encrypts the data using the client's public key and sends the encrypted data.
3. Client receives this data and decrypts it.

Since this is asymmetric, nobody else except browser can decrypt the data even if a third party has public key of browser.

The idea

The idea of RSA is based on the fact that it is difficult to factorize a large integer. The public key consists of two numbers where one number is multiplication of two large prime numbers. And private key is also derived from the same two prime numbers. So if somebody can factorize the large number, the private key is compromised. Therefore encryption strength totally lies on the key size and if we double or triple the key size, the strength of encryption increases exponentially. RSA keys can be typically 1024 or 2048 bits long, but experts believe that 1024 bit keys could be broken in the near future. But till now it seems to be an infeasible task.

Mechanism

Let us learn the mechanism behind RSA algorithm :

>> Generating Public Key :

- Select two prime no's. Suppose P = 53 and Q = 59.
- Now First part of the Public key : n = P*Q = 3127.
- We also need a small exponent say e :
- But e - Must be an integer. Not be a factor of n.
- 1 < e < Φ(n) [Φ(n) is discussed below],
- Let us now consider it to be equal to 3.

- Our Public Key is made of n and e

>> Generating Private Key :

- We need to calculate Φ(n) :
- Such that Φ(n) = (P-1)(Q-1)
- So, Φ(n) = 3016

- Now calculate Private Key, d :
- d = (k*Φ(n) + 1) / e for some integer k
- For k = 2, value of d is 2011.

Now we are ready with our – Public Key (n = 3127 and e = 3) and Private Key(d = 2011)

Now we will encrypt “HI” :

- Convert letters to numbers : H = 8 and I = 9

- Thus Encrypted Data c = 89^e mod n.
- Thus our Encrypted Data comes out to be 1394

Now we will decrypt 1394 :

- Decrypted Data = c^d mod n.

● Thus our Encrypted Data comes out to be 89

8 = H and I = 9 i.e. "HI".

Algorithm

Generating the keys -

1. Select prime numbers p,q.
2. Calculate n=p*q.
3. Calculate Φ(n)=(p-1)(q-1).
4. Select e(public key), such that 1<e<Φ(n) AND gcd(Φ(n),e)=1.
5. Calculate d(private key), such that (e*d mod Φ(n))=1.
6. Encrypt message M using
   C = M**e mod n
7. Decrypt encoded message C using
   M= C**d mod n

CODE

```
#RSA
def gcd(x,y):
	if x>y:
		small=y
	else:
		small=x
	for i in range(1,small+1):
		if x%i==0 and y%i==0:
			gcd=i
	return gcd

def generateKeys():
	p=int(input('Enter the value of p(prime):'))
	q=int(input('Enter the value of q(prime):'))
	n=p*q
	phi=(p-1)*(q-1)
	e=0
	for i in range(2,phi):
		if gcd(phi,i)==1:
			e=i
			break
	d=0
	for i in range(1,phi):
		if (e*i)%phi==1:
			d=i
			break
	return(e,d,n)

def encryptText(pt,e,n,numflag):
	ct=''
	if numflag:
```

```
            for letter in pt:
                M=ord(letter)-49
                C=(M**e)%n
                ct+=chr(C+49)
        else:
            for letter in pt:
                M=ord(letter)-97
                C=(M**e)%n
                ct+=chr(C+97)
        print('Encrypted Text : ',ct)
        return ct

def decryptText(ct,d,n,numflag):
        pt=''
        if numflag:
            for letter in ct:
                C=ord(letter)-49
                M=(C**d)%n
                pt+=chr(M+49)
        else:
            for letter in ct:
                C=ord(letter)-97
                M=(C**d)%n
                pt+=chr(M+97)
        print('Decrypted Text : ',pt)

e,d,n=generateKeys()
pt=input('Enter the plain text to encrypt : ')
numflag=1 if pt.isnumeric() else 0
enc=encryptText(pt,e,n,numflag)
decryptText(enc,d,n,numflag)
```

OUTPUT

```
================== RESTART: G:\Gursheen\221\cryptography\P4\rsa.py =
=
Enter the value of p(prime):5
Enter the value of q(prime):11
Enter the plain text to encrypt : 2342
Encrypted Text :  29L2
Decrypted Text :  2342
>>>
================== RESTART: G:\Gursheen\221\cryptography\P4\rsa.py =
=
Enter the value of p(prime):11
Enter the value of q(prime):13
Enter the plain text to encrypt : hellofromgursheen
Encrypted Text :  g³åå m¶é ëg³³Ö
Decrypted Text :  hellofromgursheen
>>>
```