**Charotar University of Science & Technology (CHARUSAT)**

**Devang Patel Institute of Advance Technology & Research (DEPSTAR)**

**Information Security (CE348)**

**Practical Solution**

|  |  |
| --- | --- |
| **Semester:6th** | **Academic Year: 2021-22** |

|  |  |
| --- | --- |
| **Sr No** | **Aim** |
| **1.** | The "Caesar Box," or "Caesar Cipher," is one of the earliest known ciphers. Developed around 100 BC, it was used by Julius Caesar to send secret messages to his generals in the field. In the event that one of his messages got intercepted, his opponent could not read them. This obviously gave him a great strategic advantage. Caesar shifted each letter of his message few letters to the right to produce what could be called the ciphertext. The ciphertext is what the enemy would see instead of the true message. So, for example, if Caesar’s messages were written in the English alphabet, and shift by 3 then each letter “A” in the message would become a "D," the "B’s" would become "E’s," and the "X's" become "A’s." This type of cipher is appropriately called a “shift cipher.” Implement the cipher in any programming language of your choice. Perform encryption, decryption. Discuss and try some possible attacks on traditional Caesar cipher.  **Code:**  #include<bits/stdc++.h>  using namespace std;  char str[100],str1[100];  void encrypt(int key)  {  for(int i=0;i<strlen(str);i++)  {  if((int)str[i]+key>122)  {  str[i] = (char)((int)str[i]+key-26);  continue;  }  else  {str[i]=(char)((int)str[i]+key);}  }  cout<<"Encrypted String is : "<<endl;puts(str);  }  void decrypt(int key)  {  for(int i=0;i<strlen(str);i++)  {  if((int)str[i]-key<97) str1[i] = (char)((int)str[i]-key+26);  else str1[i]=(char)((int)str[i]-key);  }  cout<<"Key is : "<<key <<"\nDecrypted String is : ";puts(str1);  }  int main()  {  int key;  cout<<"Enter String : "<<endl;  gets(str);  cout<<"Enter Key : "<<endl;  cin >> key;  encrypt(key);  decrypt(key);  cout<<"Attack starts : "<<endl;  for(int i=0;i<26;i++)  {  decrypt(i);  }  cout<<endl<<"Made by: Manan Patel (19DCE104) "<<endl;  return 0;  }  **Output :** |
| **2.** | The Playfair cipher was predominantly used by British forces during the Second Boer War (1899-1902) and World War I (1914-1918). Soldier from field wants to send message to base. Implement the cipher to encrypt and decrypt message.  Encrypt message : Hiroshima  Use key : pearlharbour  **Code:**  #include<bits/stdc++.h>  using namespace std;  char grid[5][5];  char keyword[26];  char msg[100];  int mark[130],len,r,c;  void createGrid();  void showGrid();  void encipher();  void decipher();  void menu()  {  int n;  string op[]={"1. Encipher","2. Decipher","3. Exit"};  cout<<op[0]<<endl<<op[1]<<endl<<op[2]<<endl<<"Enter choice: ";  cin>>n;  if(n==1) encipher();  else if(n==2) decipher();  else {cout<<endl<<"Made by: Manan Patel (19DCE104) "<<endl;exit(1);}  }  int main()  {  menu();  return 0;  }  void decipher()  {  createGrid();  showGrid();  cout<<"Cypher text: ";  char cypText[150];  cin>>cypText;  int l=strlen(cypText);  cout<<"Decipher text: ";  int P,Q,R,S,f1,f2;  char x,y;  for(int i=0;i<l;i+=2)  {  x=cypText[i];  y=cypText[i+1];  f1=f2=0;  for(int j=0;j<5;j++)  {  for(int k=0;k<5;k++)  {  if(x==grid[j][k])  {  P=j;  Q=k;  f1=1;  }  if(y==grid[j][k])  {  R=j;  S=k;  f2=1;  }  if(f1 && f2) break;  }  if(f1 && f2) break;  }  if(P==R) //same row  {  if(Q==0) cout<<grid[P][4];  else cout<<grid[P][Q-1];  if(S==0) cout<<grid[R][4];  else cout<<grid[R][S-1];  }  else if(Q==S )  {  if(P==0) cout<<grid[4][Q];  else cout<<grid[P-1][Q];  if(R==0) cout<<grid[4][S];  else cout<<grid[R-1][S];  }  else  {  cout<<grid[P][S]<<grid[R][Q];  }  }  cout<<endl<<endl;  menu();  }  void encipher()  {  createGrid();  showGrid();  cout<<"Message to cypher: ";  gets(msg);  int l=strlen(msg);  char reqText[150];  int in=0,j=0;  for(int i=0;i<l;i++)  {  j=i+1;  if(msg[i]==' ')  {  i++;  j++;  }  if(msg[j]==' ') j++;  if(toupper(msg[i])=='J') msg[i]='i';  if(toupper(msg[i])==toupper(msg[j]))  {  reqText[in]=toupper(msg[i]);  reqText[in+1]='X';  in++;  }  else  {  reqText[in]=toupper(msg[i]);  }  in++;  }  if(in%2!=0) reqText[in]='X';  cout<<"Cypher text: ";  int P,Q,R,S,f1,f2;  char x,y;  for(int i=0;i<in;i+=2)  {  x=reqText[i];  y=reqText[i+1];  f1=f2=0;  for(int j=0;j<5;j++)  {  for(int k=0;k<5;k++)  {  if(x==grid[j][k])  {  P=j;  Q=k;  f1=1;  }  if(y==grid[j][k])  {  R=j;  S=k;  f2=1;  }  if(f1 && f2) break;  }  if(f1 && f2) break;  }  if(P==R)  {  if(Q==4) cout<<grid[P][0];  else cout<<grid[P][Q+1];  if(S==4) cout<<grid[R][0];  else cout<<grid[R][S+1];  }  else if(Q==S )  {  if(P==4) cout<<grid[0][Q];  else cout<<grid[P+1][Q];  if(R==4) cout<<grid[0][S];  else cout<<grid[R+1][S];  }  else  {  cout<<grid[P][S]<<grid[R][Q];  }  }  cout<<endl<<endl;  menu();  }  void createGrid()  {  cout<<"Keyword: ";  cin>>keyword;  getchar();  len=strlen(keyword);  mark['J']=1;  r=0,c=0;  for(int i=0;i<len;i++)  {  if(!mark[toupper(keyword[i])])  {  mark[toupper(keyword[i])]=1;  grid[r][c++]=toupper(keyword[i]);  if(c%5==0)  {  c=0;  r++;  }  }  }  for(int i='A';i<='Z';i++)  {  if(mark[i]==0)  {  grid[r][c++]=i;  mark[i]=1;  if(c%5==0)  {  if(r==4 && c==5) break;  r++;  c=0;  }  }  }  }  void showGrid()  {  cout<<"5x5 Matrix"<<endl;  for(int i=0;i<5;i++)  {  for(int j=0;j<5;j++)  {  cout<<grid[i][j]<<" ";  }  cout<<endl;  }  }  **Output :** |

|  |  |
| --- | --- |
| **3.** | The Rail Fence Cipher was invented in ancient times. It was used by the Greeks, who created a special tool, called scytale, to make message encryption and decryption easier. The letters are arranged in a way which is similar to the shape of the top edge of the rail fence. If king Leonidas want to sent message to Sparta as “300 achieved glory at hot gate, unite for Greece ” then what will be ciphertext when it is encrypted using 3 rows. Also implement decryption of message.  **Code:**  #include <bits/stdc++.h>  using namespace std;  string s ={}, s1={};  int key;  void encryption(){  cout<<endl<<"--------encryption---------"<<endl;  int len;  cout<<"Enter Text :- ";  getline (cin,s);  len=s.length();  cout<<"Enter Key :- ";  cin>>key;  char ans[key][len]={};  for(int i = 0; i < key; ++i)  {for(int j = 0; j < len; ++j)  {ans[i][j] = '\n';}}  int j=-1,row=0;  for(int i=0;i<len;i++)  {  if(row==0 || row==key-1){  j=j\*(-1);  }  ans[row][i] = s[i];  row=row+j;  }  for(int i=0;i<key;i++)  {  for (int j = 0; j < len; j++) {  if(ans[i][j]!='\n'){  // cout<<ans[i][j];  s1+=ans[i][j];  }  else{ans[i][j]='-';}  }  }  cout<<endl;  for(int i=0;i<key;i++)  {  for (int j = 0; j < len; j++) {  cout<<ans[i][j];  }  cout<<endl;  }  cout<<endl;  cout<<"encryption message : - ";  cout<<s1;  }  void decryption(){  cout<<endl<<"--------decryption---------"<<endl;  int len,m=0;  len=s1.length();  char ans[key][len]={};  for(int i = 0; i < key; ++i)  {for(int j = 0; j < len; ++j)  {ans[i][j] = '\n';}}  int j=-1,row=0;  for(int i=0;i<len;i++)  {  if(row==0 || row==key-1){  j=j\*(-1);  }  ans[row][i] = '\*';  row=row+j;  }  cout<<endl;  for(int i=0;i<key;i++)  {  for (int j = 0; j < len; j++) {  if(ans[i][j]=='\*'){ans[i][j]=s1[m++];}  }  }  for(int i=0;i<key;i++)  {  for (int j = 0; j < len; j++) {  if(ans[i][j]=='\n'){cout<<"-";}  else {cout<<ans[i][j];}  }  cout<<endl;  }  cout<<endl;  cout<<"decryption message : - ";  row =0, j=-1;  for(int i = 0; i < len; i++){  if(row == 0 || row == key-1)  {j= j \* (-1);}  cout<<ans[row][i];  row = row + j;  }  }  int main()  {  encryption();  decryption();  cout<<endl<<"Made by: Manan Patel (19DCE104) "<<endl;  return 0;  }  **Output :** |

|  |  |
| --- | --- |
| **4.** | Sergio wants to pass encrypted message to Rafael. He is using Hill cipher. Message : family Key : consider 3x3 matrix Implement encryption and decryption of message.  **Code:**  #include<bits/stdc++.h>  using namespace std;  int keyMatrix[3][3], messageMatrix[3][1], msg[3][1], cipherMatrix[3][1], key\_inv[3][3];  string message;  float mi;  void encrypt();  void decrypt();  void getMatrix(){  int i, j;  cout<<"Enter 3x3 matrix for key (should have inverse):\n";  for(i = 0; i < 3; i++){  for(j = 0; j < 3; j++)  {  cin>>keyMatrix[i][j];  }  }  encrypt();  }  void encrypt()  {  cout<<"\nEnter a string of 3 letter (use A to Z): ";  cin>>message;  for(int i = 0; i < 3; i++)  {  msg[i][0] = message[i] - 65;  }  int x, i=0, j,k=0,m=0;  for (int i = 0; i < 3; i++)  {  for (x = 0; x < 3; x++)  {  cipherMatrix[i][0] +=keyMatrix[i][x] \* msg[x][0];  }  cipherMatrix[i][0] = cipherMatrix[i][0] % 26;  }  cout<<"Cipher text =";  for (int l = 0; l < 3; l++)  {  cout<<(char)(cipherMatrix[l][0]+65);  }  decrypt();  }  void inverse() {  int i, j;  int determinant = 0;  //finding determinant  for(i = 0; i < 3; i++)  {  determinant = determinant + (keyMatrix[0][i] \* (keyMatrix[1][(i+1)%3] \* keyMatrix[2][(i+2)%3] - keyMatrix[1][(i+2)%3] \* keyMatrix[2][(i+1)%3]));  }  cout<<"\ninverse key matrix"<<endl;  for(i = 0; i < 3; i++){  for(j = 0; j < 3; j++)  {key\_inv[i][j]=((keyMatrix[(j+1)%3][(i+1)%3] \* keyMatrix[(j+2)%3][(i+2)%3]) - (keyMatrix[(j+1)%3][(i+2)%3] \* keyMatrix[(j+2)%3][(i+1)%3]));  key\_inv[i][j]=(key\_inv[i][j]\*determinant)%26;  if(key\_inv[i][j]<0) {key\_inv[i][j] = 26-(key\_inv[i][j]\*(-1));}  }  }  cout<<endl;  for(i = 0; i < 3; i++){  for(j = 0; j < 3; j++)  {  cout<<key\_inv[i][j]<<" ";  }  cout<<endl;  }  }  void decrypt(){  inverse();  for (int i = 0; i < 3; i++)  {  for (int x = 0; x < 3; x++)  {  messageMatrix[i][0] +=key\_inv[i][x] \* cipherMatrix[x][0];  }  messageMatrix[i][0] = messageMatrix[i][0] % 26;  }  cout<<endl<<"Decrypt message = ";  for (int l = 0; l < 3; l++)  {  cout<<(char)(messageMatrix[l][0]+65);  }  }  int main()  {  getMatrix();  cout<<endl<<"Made by: Manan Patel (19DCE104) "<<endl;  return 0;  }  **Output :** |

|  |  |
| --- | --- |
| **5.** | Mr. Lucious Fox wants to transfer small amount of data within one session to Bruce wayne. But they know that joker is listening/tapping to communication so they want communication to be encrypted with secret key. Implement Diffie hellman algorithm to help them establishing key for session.  **Code:**  #include<math.h>  #include<iostream>  using namespace std;  int main()  {  int p,g,x,y;  cout<<"Enter value of p , g : ";  cin >>p>>g;  cout<<"Enter private key a for Lucious Fox : ";  cin>>x;  cout<<"Enter private key a for Bruce wayne : ";  cin>>y;  int a = pow(g,x);a=a%p;  int b = pow(g,y);b=b%p;  int ska =pow(b,x);  ska = ska%p;  int skb =pow(a,y);  skb = skb%p;  cout<<"\nSecret key for Lucious Fox is "<<ska<<"\nSecret key for Bruce wayne is "<<skb<<endl;  cout<<endl<<"Made by: Manan Patel (19DCE104) "<<endl;  return 0;  }  **Output :** |

|  |  |
| --- | --- |
| **6.** | After establishing connection with bruce wayne , established shared secret is used as a input to a random number generator available at both ends. Generated random numbers will follow same sequence at both ends. They are used as a one time pad for encrypting/decrypting message. Message is converted to binary numbers and then encrypted with ex-or operation. Implement above system as a stream of message. Consider A=1, B=2, C=0…. So one.  **Code:**  import random  def generate\_key(msg):  key = list()  for i in range(len(msg)):  key.append(random.randint(0, 25))  return tuple(key)  def encrypt(msg, key):  cipher\_text = str()  for i in range(len(msg)):  cipher\_text = cipher\_text + chr(((ord(msg[i]) - ord('a')) + key[i]) % 26 + ord('a'))  return cipher\_text  def decrypt(cipher\_text, key):  msg = str()  for i in range(len(cipher\_text)):  msg = msg + chr(((ord(cipher\_text[i]) - ord('a')) - key[i]) % 26 + ord('a'))  return msg  msg = input("Enter message to send : ")  key = generate\_key(msg)  cipher\_text = encrypt(msg, key)  print("\nEncrypted message : ", cipher\_text)  decrypted = decrypt(cipher\_text, key)  print("Decrypted message : ", decrypted)  print("\nKeys used : ", key)  **Output :** |

|  |  |
| --- | --- |
| **7.** | RSA algorithm is used by Salim to transfer session key to Anarkali. He suspects that Akbar is performing man in middle attack he chose to use 1024 bit prime numbers. Hint: you may choose to use big integer.  **Code:**  import math  def checkPrime(n):  k = math.sqrt(n)  for i in range(3,int(k+1)):  if n%i == 0:  return False  return True  def gcd(a,b):  if b == 0:  return a  return gcd(b,a%b)  def get\_public\_key(phi):  for i in range(2,10000):  e = gcd(i,phi)  if e == 1:  return i  return False  def get\_private\_key(phi, e, t1, t2):  global d  if e == 0:  d = t1  return t1  get\_private\_key(e, phi%e, t2, t1-(phi//e)\*t2)  def encrpyt(M, e, n):  return (M\*\*e)%n  def decrypt(C, d, n):  return (C\*\*d)%n  p = int(input("Enter First Prime number: "))  while not checkPrime(p):  print(f'{p} is not a prime number. Try Again.')  p = int(input("\nEnter First Prime number: "))  q = int(input("\nEnter Second Prime number: "))  while not checkPrime(q):  print(f'{q} is not a prime number. Try Again.')  q = int(input("\nEnter Second Prime number: "))  n = p\*q  phi = (p-1)\*(q-1)  e = get\_public\_key(phi)  if e:  get\_private\_key(phi, e, 0, 1)  if d<0:  d = d+phi  print(f'\nPublic ckey e = {e}, Private key d = {d}')  M = int(input("\nEnter Your Message (Number): "))  C = encrpyt(M, e, n)  print(f"\nEncrypted Message: {C}")  original\_message = decrypt(C, d, n)  print(f"\nDecrypted Message: {original\_message}")  print("\nMade by: Manan Patel (19DCE104)")  **Output :** |

|  |  |
| --- | --- |
| **8.** | Salim received some doubtful message from Anarkali. He is doubtful that messages are tempered by some attacker during transmission so he asks Anarkali to digitally sign the messages. Implement system for transferring message with non-repudiation.  **Code:**  e1,d1,n1 = 5,29,91 # assuming the public & private key for anarkali  # e = public key, d = private key  # now if anarkali wants to talk to salim & if salim wants to also verify that message was sent by anarkali,  # anarkali should digitally sign the message with her private key.  # salim should decrypt the message with her public key.  # this way attacker can't modify the message.  message\_1 = int(input("Enter person1's Message:"))  # encrypt  def encrypt(msg):  # encrypting with 1's private key  return pow(msg, d1, n1)  #decrypt  def decrypt(cipher):  # decrypting with 1's public key  return pow(cipher, e1, n1)  cipher = encrypt(message\_1)  print(f"Cipher Text:{cipher}")  pt = decrypt(cipher)  if (message\_1 == pt):  print("Message Verified!")  print(f"Plain Text:{pt}")  **Output :** |
| **9.** | Viru sent important content in file along with hash of file content along with it. At the receivers’ end, how Raju can check integrity of file? Show that even slight in change in file content will bring significant change in hash value.  **Code:**  import hashlib  import random  # Writing a file  with open("hash.txt", 'w') as f:  f.write("You can never understand everything. But, you should push yourself to understand the system! -Ryan Dahl")  # lets suppose viru sends above file hash.txt to raju along with the hash  with open("hash.txt", 'r') as f:  msg\_1 = f.read()  hash = hashlib.sha1(msg\_1.encode())  # print(hash.\_\_hash\_\_())  def alter\_msg(msg):  r = random.randint(0, 1)  msg\_1, hash = msg  if r:  msg\_1 += str(r) # alter the string by 1 char at random  print(msg\_1)  return (msg\_1, hash)  def check\_msg(message):  x, y = message  a = hashlib.sha1(str(x).encode()).hexdigest()  b = y.hexdigest()  if a == b:  print("The message is not altered!")  else:  print("The message is altered!")  msg\_o = (msg\_1, hash) # a tuple of msg along with it's hash  message = alter\_msg(msg\_o) # a random function to alter the message  check\_msg(message) # to check is msg is altered or not  **Output :** |

|  |  |
| --- | --- |
| **10.** | Demonstrate working of Digital Signature using Cryptool.  **Output:**      Now Click on hash function & generate hash    Now click on generate key & generate key    Now click on provide certificate, generate certificate & store the signature & thus our required digital signature is generated |

|  |  |
| --- | --- |
| **11.** | Demonstrate image steganography using any programming language of your choice.  **Code:**  %Encoding the message  original=imread('762226-top-beautiful-3d-wallpaper-1920x1200-for-full-hd.jpg');  cover=rgb2gray(original);  [row,column]=size(cover);  L=256;  stego=cover;  message=input('Enter the message to be hidden: ','s');  len=strlength(message)\*8; %Each character will take 8 bits so total number of bits in the message will be len  ascii=uint8(message); %ascii is a vector having the ascii value of each character  binary\_separate=dec2bin(ascii,8); %binary\_separate is an array having the decimal representation of each ascii value  binary\_all=''; %binary\_all will have the entire sequence of bits of the message  for i=1:strlength(message)  binary\_all=append(binary\_all,binary\_separate(i,:));  end  count=1; %initializing count with 1  for i=1:row  for j=1:column  %for every character in the message  if count<=len  %Obtain the LSB of the grey level of the pixel  LSB=mod(cover(i,j),2);    %Convert the bit from the message to numeric form  a=str2double(binary\_all(count));    %Perform XOR operation between the bit and the LSB  temp=double(xor(LSB,a));    %Change the bit of the stego image accordingly  stego(i,j)=cover(i,j)+temp;    count=count+1;  end  end  end  subplot(1,2,1);  imshow(cover);  title('Cover Image');  subplot(1,2,2);  imshow(stego);  title('Stego Image');  %Decoding the message  count=1;  message\_in\_bits='';  for i=1:row  for j=1:column  %For all the characters in the message  if count<=len    %Retrieve the LSB of the intensity level of the pixel  LSB=mod(stego(i,j),2);    %Append into message\_in\_bits to get bit sequence of message  message\_in\_bits=append(message\_in\_bits,num2str(LSB));    count=count+1;  end  end  end  %Converting the bit sequence into the original message  i=1;  original\_message='';  while i<=len  %Take a set of 8 bits at a time  %Convert the set of bits to a decimal number  %Convert the decimal number which is the ascii value to its corresponding character  %Append the obtained character into the resultant string    original\_message=append(original\_message,char(bin2dec(message\_in\_bits(1,i:i+7))));  i=i+8;  end  disp(['The original message is: ',original\_message]);  **Output:** |

|  |  |
| --- | --- |
| **12.** | Implement blockchain based protocol considering any application of your choice (for example e-voting, fake document detection etc.)  **Code:**  pragma solidity 0.4.23;  contract Election {  // Model a Candidate  struct Candidate {  uint id;  string name;  uintvoteCount;  }  }  // Store accounts that have voted  mapping(address => bool)  public voters;  // Store Candidates  // Fetch Candidate  mapping(uint => Candidate)  public candidates;  // Store Candidates Count  uint public candidatesCount;  // voted event  event votedEvent (uint indexed \_candidateId);  constructor () public {  addCandidate("Candidate1");  addCandidate("Candidate2");  }  function addCandidate (string \_name)  private {  candidatesCount ++;  candidates[candidatesCount] = Candidate(candidatesCount, \_name, 0);  }  function vote (uint \_candidateId) public {  // require that they haven't voted before  require(!voters[msg.sender]);  // require a valid candidate  require(\_candidateId> 0 && \_candidateId<= candidatesCount);  // record that voter has voted  voters[msg.sender] = true;  // update candidate vote Count  candidates[\_candidateId].voteCount ++;  // trigger voted event  emit votedEvent(\_candidateId);  } |