Practical-10

Aim : Implement encryption and decryption using Simplified-DES scheme

Code :

```
#include<stdio.h>
#include<string.h>
int s_matrix0[4][4] = {   {1,0,3,2},
{3,2,1,0},
{0,2,1,3},
{3,1,3,2} };
int s_matrix1[4][4] = { {0,1,2,3},
{2,0,1,3},
{3,0,1,0},
{2,1,0,3} };
int s0=0, s1=0; int row=0, col=0;
int s0_binary[2], s1_binary[2]; int result[2];
int to_digit(int a, int b){
int output; if(a==1 && b==1)
output = 3;
if(a==0 && b==1) output = 1;
if(a==1 && b==0) output = 2;
if(a==0 && b==0) output = 0;
return output;
}
void to_binary(int num){
int x;
if(num == 3){
for(x=0; x<2; x++) result[x] = 1;
}
else if(num == 1){
result[0] = 0;
```

```
result[1] = 1;
}
else if(num == 2){
result[0] = 1;
result[1] = 0;
}
else if(num == 0){
for(x=0; x<2; x++) result[x] = 0;
}
}
void main(){
int i,j,index;
int k1[8], k2[8];
int afterp10[11], ls1[10], ls2[10], afterip[8], afterep[8], afterp4[4]; int aftersboxesone[4], aftersboxestwo[4];
int leftafterip[4], rightafterip[4]; int leftafterep[4], rightafterep[4];
int leftafterone[4], rightafterone[4]; int afteripinverse[8];
int afterone[8], aftertwo[8];
int p10[10] = {3,5,2,7,4,10,1,9,8,6};
int p8[8] = {6,3,7,4,8,5,10,9};
int ip[8] = {2,6,3,1,4,8,5,7};
int ep[8] = {4,1,2,3,2,3,4,1}; int p4[4] = {2,4,3,1};
int ipinverse[8] = {4,1,3,5,7,2,8,6};
int key[11] = {1,0,1,0,0,0,0,0,1,0};
int plain[8] = {0,1,1,0,1,1,0,1};
printf("S-DES Key Generation and Encryption.\n");
printf("\n----------KEY GENERATION \n");
printf("\nEntered the 10-bit key is: ");
for(i=0; i<10; i++) printf("%d ",key[i]);
printf("\np10 permutation is defined as: ");
for(i=0; i<=9; i++) printf("%d ",p10[i]);
for(i=0; i<=9; i++){
```

```
index = p10[i];
afterp10[i] = key[index - 1];
}
afterp10[i] = '\0';
printf("\n\nAfter p10 = ");
for(i=0; i<=9; i++) printf("%d ",afterp10[i]);
for(i=0; i<5; i++){
if(i == 4)
ls1[i]=afterp10[0];
else
ls1[i]=afterp10[i+1];
}
for(i=5; i<10; i++){
if(i == 9)
ls1[i]=afterp10[5];
else
ls1[i]=afterp10[i+1];
}
printf("\n\nAfter LS-1 = ");
 for(i=0; i<10; i++) printf("%d ", ls1[i]);
printf("\np8 permutation is defined as: ");
for(i=0;i<8;i++) printf("%d ",p8[i]);
index=0;
for(i=0; i<=9; i++){
index = p8[i];
k1[i] = ls1[index - 1];
}
printf("\n\n  >Key-1 is: ");
for(i=0;i<8;i++) printf("%d ",k1[i]);
for(i=0; i<3; i++){
ls2[i]=ls1[i+2];
}
```

```
ls2[3]=ls1[0];
ls2[4]=ls1[1];
for(i=5; i<8; i++){
ls2[i]=ls1[i+2];
}
ls2[8]=ls1[5];
ls2[9]=ls1[6];
printf("\n\nAfter LS-2 = ");
for(i=0; i<10; i++) printf("%d ", ls2[i]);
printf("\np8 permutation is defined as: ");
for(i=0;i<8;i++) printf("%d ",p8[i]);
index=0;
for(i=0; i<=9; i++){
index = p8[i];
k2[i] = ls2[index - 1];
}
printf("\n\n  >Key-2 is: ");
for(i=0;i<8;i++) printf("%d ",k2[i]);
printf("\n\n----------S-DES ENCRYPTION     \n");
printf("\n      >Entered the 8-bit plaintext is: ");
for(i=0; i<8; i++) printf("%d ",plain[i]);
printf("\nInitial permutation is defined as: ");
for(i=0; i<8; i++) printf("%d ",ip[i]);
for(i=0; i<8; i++){
index = ip[i];
afterip[i] = plain[index - 1];
}
afterip[i] = '\0';
printf("\nAfter ip = ");
for(i=0; i<8; i++) printf("%d ", afterip[i]);
printf("\n\nExpand permutation is defined as: ");
for(i=0; i<8; i++) printf("%d ",ep[i]);
```

```
for(j=0; j<4; j++) leftafterip[j] = afterip[j];
for(j=0; j<4; j++) rightafterip[j] = afterip[j+4];
for(i=0; i<4; i++){
index = ep[i];
afterep[i] = rightafterip[index - 1];
}
for(i=4; i<8; i++){
index = ep[i];
afterep[i] = rightafterip[index - 1];
}
afterep[i] = '\0';
printf("\nAfter ep = ");
for(i=0; i<8; i++) printf("%d ", afterep[i]);
for(i=0; i<8; i++) k1[i] = k1[i] ^ afterep[i];
printf("\n\nAfter XOR operation with 1st Key= ");
for(i=0; i<8; i++) printf("%d ", k1[i]);
row = to_digit(k1[0],k1[3]);
col = to_digit(k1[1],k1[2]);
s0 = s_matrix0[row][col];
to_binary(s0);
for(j=0; j<2; j++) s0_binary[j] = result[j];
row = to_digit(k1[4],k1[7]);
col = to_digit(k1[5],k1[6]);
s1 = s_matrix1[row][col];
to_binary(s1);
for(j=0; j<2; j++) s1_binary[j] = result[j];
for(j=0; j<2; j++) aftersboxesone[j] = s0_binary[j];
for(i=0,j=2; i<2,j<4; i++,j++) aftersboxesone[j] = s1_binary[i];
printf("\n\nAfter first S-Boxes= ");
for(i=0; i<4; i++) printf("%d ", aftersboxesone[i]);
printf("\n\nP4 is defined as: ");
for(i=0; i<4; i++) printf("%d ",p4[i]);
```

```
for(i=0; i<4; i++){
index = p4[i];
afterp4[i] = aftersboxesone[index - 1];
}
afterp4[i] = '\0';
printf("\nAfter P4= ");
for(i=0; i<4; i++) printf("%d ", afterp4[i]);
for(i=0; i<4; i++) afterp4[i] = afterp4[i] ^ leftafterip[i];
printf("\n\nAfter XOR operation with left nibble of after ip= ");
for(i=0; i<4; i++) printf("%d ", afterp4[i]);
for(i=0; i<4; i++) afterone[i] = rightafterip[i];
for(i=0,j=4; i<4,j<8; i++,j++) afterone[j] = afterp4[i];
afterone[j] = '\0';
printf("\nAfter first part= ");
for(i=0; i<8; i++) printf("%d ", afterone[i]);
for(j=0; j<4; j++) leftafterone[j] = afterone[j];
for(j=0; j<4; j++) rightafterone[j] = afterone[j+4]
printf("\n\nExpand permutation is defined as: ");
for(i=0; i<8; i++) printf("%d ",ep[i]);
for(i=0; i<4; i++){
index = ep[i];
afterep[i] = rightafterone[index - 1];
for(i=4; i<8; i++){
index = ep[i];
afterep[i] = rightafterone[index - 1];
}
afterep[i] = '\0';
printf("\nAfter second ep = ");
for(i=0; i<8; i++) printf("%d ", afterep[i]);
for(i=0; i<8; i++) k2[i] = k2[i] ^ afterep[i];
printf("\n\nAfter XOR operation with 2nd Key= ");
for(i=0; i<8; i++) printf("%d ", k2[i]);
```

```
row = to_digit(k2[0],k2[3]);
col = to_digit(k2[1],k2[2]);
s0 = s_matrix0[row][col]; to_binary(s0);
for(j=0; j<2; j++) s0_binary[j] = result[j];
row = to_digit(k2[4],k2[7]);
col = to_digit(k2[5],k2[6]);
s1 = s_matrix1[row][col];
to_binary(s1);
for(j=0; j<2; j++) s1_binary[j] = result[j];
for(j=0; j<2; j++) aftersboxestwo[j] = s0_binary[j];
 for(i=0,j=2; i<2,j<4; i++,j++) aftersboxestwo[j] = s1_binary[i];
printf("\n\nAfter second S-Boxes= ");
 for(i=0; i<4; i++) printf("%d ", aftersboxestwo[i]);
printf("\n\nP4 is defined as: ");
for(i=0; i<4; i++) printf("%d ",p4[i]);
for(i=0; i<4; i++){
index = p4[i];
afterp4[i] = aftersboxestwo[index - 1];
}
afterp4[i] = '\0';
printf("\nAfter P4= ");
 for(i=0; i<4; i++) printf("%d ", afterp4[i]);
for(i=0; i<4; i++) afterp4[i] = afterp4[i] ^ leftafterone[i];
printf("\n\nAfter XOR operation with left nibble of after first part= ");
for(i=0; i<4; i++) printf("%d ", afterp4[i]);
for(i=0; i<4; i++) aftertwo[i] = afterp4[i];
for(i=0,j=4; i<4,j<8; i++,j++) aftertwo[j] = rightafterone[i];
aftertwo[j] = '\0';
printf("\n\nAfter second part= ");
 for(i=0; i<8; i++) printf("%d ", aftertwo[i]);
printf("\n\nInverse Initial permutation is defined as: ");
 for(i=0; i<8; i++) printf("%d ",ipinverse[i])
```

```
for(i=0; i<8; i++){
index = ipinverse[i];
afteripinverse[i] = aftertwo[index - 1];
}
afteripinverse[j] = '\0';
printf("\n\n--->8-bit Ciphertext will be= ");
 for(i=0; i<8; i++) printf("%d ", afteripinverse[i]);
}
```

Output :

```
E:\College ppts\SEMESTER 6\CNS PRACTICALS\DES.exe
S-DES Key Generation and Encryption.

----------KEY GENERATION----------

Entered the 10-bit key is: 1 0 1 0 0 0 0 0 1 0
p10 permutation is defined as: 3 5 2 7 4 10 1 9 8 6

After p10 = 1 0 0 0 0 0 1 1 0 0

After LS-1 = 0 0 0 0 1 1 1 0 0 0
p8 permutation is defined as: 6 3 7 4 8 5 10 9

---->Key-1 is: 1 0 1 0 0 1 0 0

After LS-2 = 0 0 1 0 0 0 0 0 1 1
p8 permutation is defined as: 6 3 7 4 8 5 10 9

---->Key-2 is: 0 1 0 0 0 0 1 1

----------S-DES ENCRYPTION---------

---->Entered the 8-bit plaintext is: 0 1 1 0 1 1 0 1
Initial permutation is defined as: 2 6 3 1 4 8 5 7
After ip = 1 1 1 0 0 1 1 0

Expand permutation is defined as: 4 1 2 3 2 3 4 1
After ep = 0 0 1 1 1 1 0 0

After XOR operation with 1st Key= 1 0 0 1 1 0 0 0

After first S-Boxes= 1 1 1 1

P4 is defined as: 2 4 3 1
After P4= 1 1 1 1

After XOR operation with left nibble of after ip= 0 0 0 1
After first part= 0 1 1 0 0 0 0 1

Expand permutation is defined as: 4 1 2 3 2 3 4 1
After second ep = 1 0 0 0 0 0 1 0

After XOR operation with 2nd Key= 1 1 0 0 0 0 0 1

After second S-Boxes= 0 1 1 0

P4 is defined as: 2 4 3 1
After P4= 1 0 1 0

After XOR operation with left nibble of after first part= 1 1 0 0
```

```
E:\College ppts\SEMESTER 6\CNS PRACTICALS\DES.exe
S-DES Key Generation and Encryption.

----------KEY GENERATION----------

Entered the 10-bit key is: 1 0 1 0 0 0 0 0 1 0
p10 permutation is defined as: 3 5 2 7 4 10 1 9 8 6

After p10 = 1 0 0 0 0 0 1 1 0 0

After LS-1 = 0 0 0 0 1 1 1 0 0 0
p8 permutation is defined as: 6 3 7 4 8 5 10 9

---->Key-1 is: 1 0 1 0 0 1 0 0

After LS-2 = 0 0 1 0 0 0 0 0 1 1
p8 permutation is defined as: 6 3 7 4 8 5 10 9

---->Key-2 is: 0 1 0 0 0 0 1 1

----------S-DES ENCRYPTION---------

---->Entered the 8-bit plaintext is: 0 1 1 0 1 1 0 1
Initial permutation is defined as: 2 6 3 1 4 8 5 7
After ip = 1 1 1 0 0 1 1 0

Expand permutation is defined as: 4 1 2 3 2 3 4 1
After ep = 0 0 1 1 1 1 0 0

After XOR operation with 1st Key= 1 0 0 1 1 0 0 0

After first S-Boxes= 1 1 1 1

P4 is defined as: 2 4 3 1
After P4= 1 1 1 1

After XOR operation with left nibble of after ip= 0 0 0 1
After first part= 0 1 1 0 0 0 0 1

Expand permutation is defined as: 4 1 2 3 2 3 4 1
After second ep = 1 0 0 0 0 0 1 0

After XOR operation with 2nd Key= 1 1 0 0 0 0 0 1

After second S-Boxes= 0 1 1 0

P4 is defined as: 2 4 3 1
After P4= 1 0 1 0

After XOR operation with left nibble of after first part= 1 1 0 0
```

```
E:\College ppts\SEMESTER 6\CNS PRACTICALS\DES.exe
After XOR operation with 1st Key= 1 0 0 1 1 0 0 0

After first S-Boxes= 1 1 1 1

P4 is defined as: 2 4 3 1
After P4= 1 1 1 1

After XOR operation with left nibble of after ip= 0 0 0 1
After first part= 0 1 1 0 0 0 0 1

Expand permutation is defined as: 4 1 2 3 2 3 4 1
After second ep = 1 0 0 0 0 0 1 0

After XOR operation with 2nd Key= 1 1 0 0 0 0 0 1

After second S-Boxes= 0 1 1 0

P4 is defined as: 2 4 3 1
After P4= 1 0 1 0

After XOR operation with left nibble of after first part= 1 1 0 0

After second part= 1 1 0 0 0 0 0 1

Inverse Initial permutation is defined as: 4 1 3 5 7 2 8 6

--->8-bit Ciphertext will be= 0 1 0 0 0 1 1 0
--------------------------------
Process exited after 0.227 seconds with return value 2
Press any key to continue . . .
```

Analysis :

Time Complexity: The time complexity of S-DES encryption and decryption is O(n^2), where n is the number of bits in the input message. This is because S-DES uses a Feistel network, which involves performing a series of operations on two halves of the message for a certain number of rounds. In S-DES, there are only two rounds, so the time complexity is relatively low.

Space Complexity: The space complexity of S-DES is also O(n^2), because the algorithm requires storing the key, the input message, and the intermediate values generated during the encryption or decryption process.

Practical-11

Aim : Implement encryption and decryption using AES scheme.

Code :

```
import javax.crypto.Cipher;
import javax.crypto.SecretKey;
import javax.crypto.SecretKeyFactory;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.PBEKeySpec;
import javax.crypto.spec.SecretKeySpec;
import java.nio.charset.StandardCharsets;
import java.security.InvalidAlgorithmParameterException;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import java.security.spec.InvalidKeySpecException;
import java.security.spec.KeySpec;
import java.util.Base64;
import javax.crypto.BadPaddingException;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.NoSuchPaddingException;
public class AESExample{
/* Private variable declaration */
private static final String SECRET_KEY = "123456789";
private static final String SALTVALUE = "abcdefg";
/* Encryption Method */
public static String encrypt(String strToEncrypt){
try{
/* Declare a byte array. */
byte[] iv = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
IvParameterSpec ivspec = new IvParameterSpec(iv);
```

```
/* Create factory for secret keys. */
SecretKeyFactory factory = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA256");
/* PBEKeySpec class implements KeySpec interface. */
KeySpec spec = new PBEKeySpec(SECRET_KEY.toCharArray(),
SALTVALUE.getBytes(), 65536, 256);
SecretKey tmp = factory.generateSecret(spec);
SecretKeySpec secretKey = new SecretKeySpec(tmp.getEncoded(),"AES");
Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
cipher.init(Cipher.ENCRYPT_MODE, secretKey, ivspec);
/* Retruns encrypted value. */
return Base64.getEncoder().encodeToString(cipher.doFinal(strToEncrypt.getBytes
(StandardCharsets.UTF_8)));
}
catch (InvalidAlgorithmParameterException | InvalidKeyException |
NoSuchAlgorithmException | InvalidKeySpecException | BadPaddingException |
IllegalBlockSizeException | NoSuchPaddingException e){
System.out.println("Error occured during encryption: " + e.toString());
}
return null;
}
/* Decryption Method */
public static String decrypt(String strToDecrypt){
try{
/* Declare a byte array. */
byte[] iv = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
IvParameterSpec ivspec = new IvParameterSpec(iv);
/* Create factory for secret keys. */
SecretKeyFactory factory = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA256");
/* PBEKeySpec class implements KeySpec interface. */
KeySpec spec = new PBEKeySpec(SECRET_KEY.toCharArray(),
SALTVALUE.getBytes(), 65536, 256);
SecretKey tmp = factory.generateSecret(spec);
```

```
SecretKeySpec secretKey = new SecretKeySpec(tmp.getEncoded(),"AES");
Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5PADDING");
cipher.init(Cipher.DECRYPT_MODE, secretKey, ivspec);
/* Retruns decrypted value. */
return new String(cipher.doFinal(Base64.getDecoder().decode(strToDecrypt)));
}
catch (InvalidAlgorithmParameterException | InvalidKeyException |
NoSuchAlgorithmException | InvalidKeySpecException | BadPaddingException |
IllegalBlockSizeException | NoSuchPaddingException e){
System.out.println("Error occured during decryption: " + e.toString());
}
return null;
}
/* Driver Code */
public static void main(String[] args){
/* Message to be encrypted. */
String originalval = "AES Encryption";
/* Call the encrypt() method and store result of encryption. */
String encryptedval = encrypt(originalval);
/* Call the decrypt() method and store result of decryption. */
String decryptedval = decrypt(encryptedval);
/* Display the original message, encrypted message and decrypted
message on the console. */
System.out.println("Original value: " + originalval);
System.out.println("Encrypted value: " + encryptedval);
System.out.println("Decrypted value: " + decryptedval);
}
}
```

Output:

Original value: AES Encryption

Encrypted value: V5E9I52IxhMaW4+hJhl56g==

Decrypted value: AES Encryption

Analysis:

Time Complexity:

The Advanced Encryption Standard (AES) is a widely used symmetric-key encryption algorithm. The time and space complexity of AES depends on various factors, such as the key size, block size, and the implementation details.

Assuming a standard implementation of AES with a block size of 128 bits and a key size of 128, 192, or 256 bits, the time complexity of the encryption and decryption process is O(N), where N is the number of blocks to be processed. For a single block, the time complexity is constant, which means that it does not depend on the input size.

Space Complexity:

The space complexity of the AES algorithm depends on the key size and the implementation details. The space required for the key schedule is proportional to the key size, which means that it is O(K), where K is the key size in bits. For the encryption and decryption process, the space complexity is constant and does not depend on the input size.

Practical-12

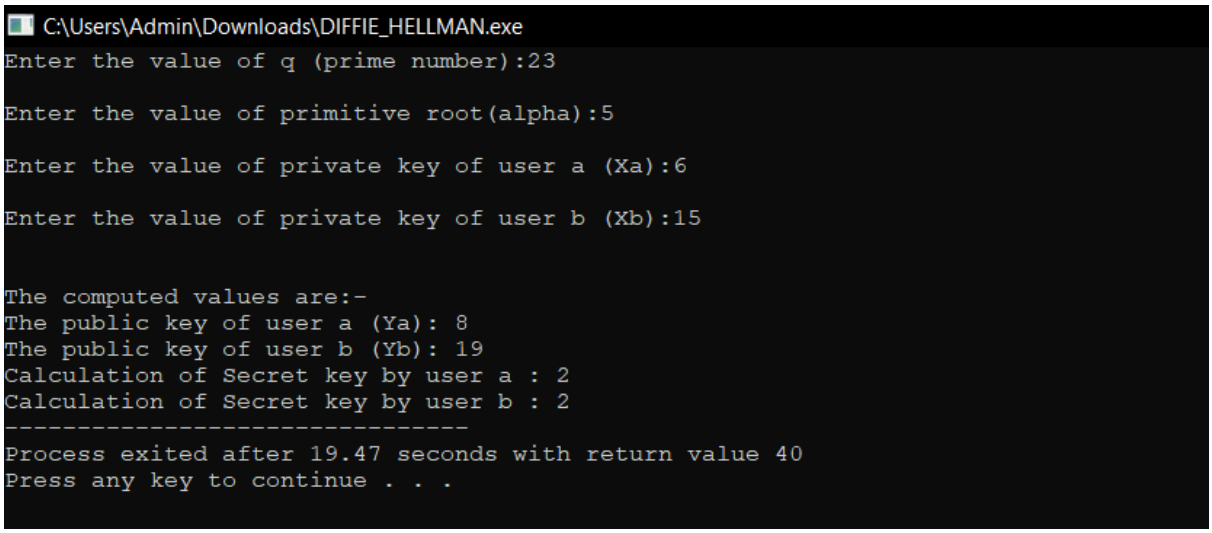
Aim : Implement Diffie-Hellman Key Exchange algorithm

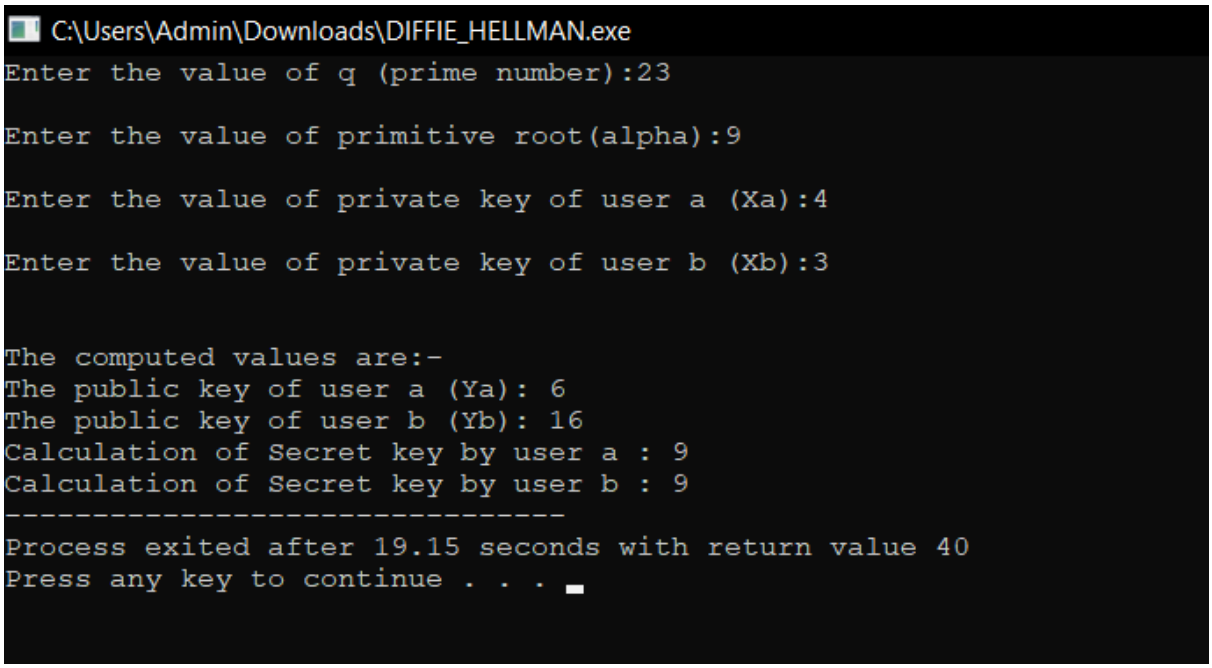
Code :

```
#include<stdio.h>
void main(){
   int q,alpha,Xa,Xb,Ya,Yb,i,pr,Ka,Kb;
   printf("Enter the value of q (prime number):");
   scanf("%d",&q);
   printf("\nEnter the value of primitive root(alpha):");
   scanf("%d",&alpha);
   printf("\nEnter the value of private key of user a (Xa):");
   scanf("%d",&Xa);
   printf("\nEnter the value of private key of user b (Xb):");
   scanf("%d",&Xb);
   printf("\n\nThe computed values are:- ");
      Ya = alpha;
      //public key of user a
      for(i=2;i<=Xa;i++){
         Ya = (Ya*alpha) % q;
      }
      printf("\nThe public key of user a (Ya): %d",Ya);
      Yb = alpha;
      //public key of user b
      for(i=2;i<=Xb;i++){
          Yb = (Yb*alpha)%q;
      printf("\nThe public key of user b (Yb): %d",Yb);
      Ka = Yb;
      for(i=2;i<=Xa;i++){
         Ka = (Ka*Yb)%q;
      }
      Kb = Ya;
      for(i=2;i<=Xb;i++){
        Kb = (Kb*Ya)%q;
```

```
    }
    printf("\nCalculation of Secret key by user a : %d",Ka);
    printf("\nCalculation of Secret key by user b : %d",Kb)
}
```

Output :





Analysis:

Time Complexity**:** The time complexity of the Diffie-Hellman key exchange algorithm is O(n^3), where n is the bit-length of the prime modulus used in the key exchange. The main computational effort in the algorithm comes from the modular exponentiation operation used in the key generation process.

Space Complexity: The space complexity of the Diffie-Hellman key exchange algorithm is O(1), meaning that the amount of memory required to execute the algorithm does not depend on the input size or the number of iterations. The algorithm only requires a constant amount of memory to store the intermediate values during the computation.