Assignment No 3

Group B (Network Security)

Aim: Implement a client and a server on different computers using python. Perform the encryption of message of sender between these two entities by using DES Algorithm and use Diffie Hellman method for exchange of keys.

Objectives:

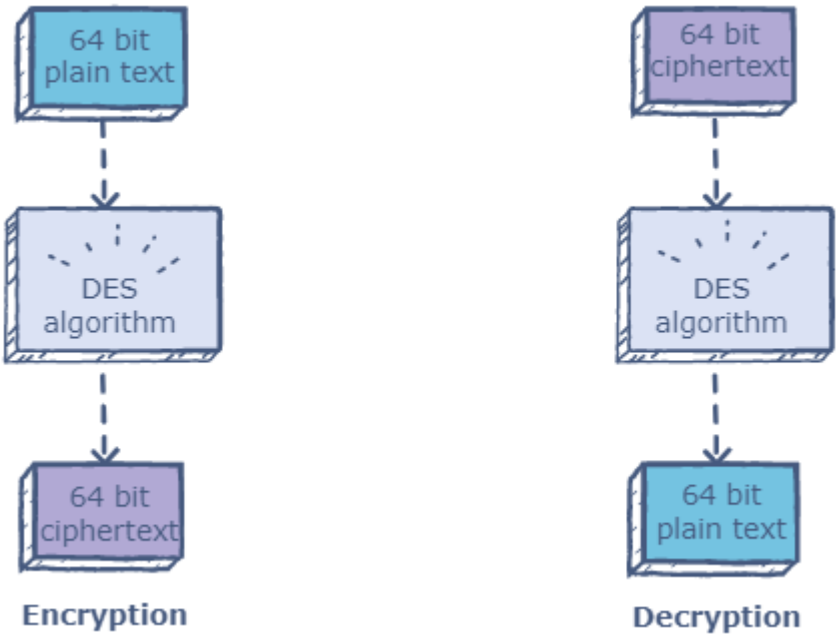
1. To learn various client/server environments to use application layer protocols.
2. To understand the network security by using public key cryptography algorithms.

Theory:

DES algorithm

Data Encryption Standard (DES) is a block cipher algorithm that takes plain text in blocks of 64 bits and converts them to ciphertext using keys of 48 bits. It is a symmetric key algorithm, which means that the same key is used for encrypting and decrypting data.

Encryption and decryption using the DES algorithm.



Steps for generating keys

There are 16 rounds of encryption in the algorithm, and a different key is used for each round. How keys are generated is listed below.

Bits are labeled from 1 to 64 starting from the most significant bit and going to the least significant bit.

1. Compress and transpose the given 64-bit key into a 48-bit key using the following table:

```
// The array elements denote the bit numbers
int pc1[56] = {
  57,49,41,33,25,17,9,
  1,58,50,42,34,26,18,
  10,2,59,51,43,35,27,
  19,11,3,60,52,44,36,
  63,55,47,39,31,23,15,
  7,62,54,46,38,30,22,
  14,6,61,53,45,37,29,
  21,13,5,28,20,12,4
};
```

2. Divide the result into two equal parts: C and D.

3. C and D are left-shifted circularly. For encryption rounds 1, 2, 9, and 16 they are left shifted circularly by 1 bit; for all of the other rounds, they are left-circularly shifted by 2.

4. The result is compressed to 48 bits in accordance with the following rule:

```
int pc2[48] = {
  14,17,11,24,1,5,
  3,28,15,6,21,10,
  23,19,12,4,26,8,
  16,7,27,20,13,2,
  41,52,31,37,47,55,
  30,40,51,45,33,48,
  44,49,39,56,34,53,
  46,42,50,36,29,32
};
```

5. The result of step 3 is the input for the next round of key generation.

Steps for encryption

1. Transpose the bits in the 64-block according to the following:

```
// 58 means that the 58th bit should be considered
// the first bit, 50th bit the second bit and so on.
int initial_permutation_table[64] = {
    58,50,42,34,26,18,10,2,
```

```
    60,52,44,36,28,20,12,4,
    62,54,46,38,30,22,14,6,
    64,56,48,40,32,24,16,8,
    57,49,41,33,25,17,9,1,
    59,51,43,35,27,19,11,3,
    61,53,45,37,29,21,13,5,
    63,55,47,39,31,23,15,7
    };
```

2. Divide the result into equal parts: left plain text (1-32 bits) and right plain text (33-64 bits)
3. The resulting parts undergo 16 rounds of encryption in each round.

The right plain text is expanded using the following expansion table:

```
// The array elements denote the bit numbers
int expansion_table[48] = {
  32,1,2,3,4,5,4,5,
  6,7,8,9,8,9,10,11,
  12,13,12,13,14,15,16,17,
  16,17,18,19,20,21,20,21,
  22,23,24,25,24,25,26,27,
  28,29,28,29,30,31,32,1
};
```

4. The expanded right plain text now consists of 48 bits and is XORed with the 48-bit key.
5. The result of the previous step is divided into 8 boxes. Each box contains 6 bits. After going through the eight substitution boxes, each box is reduced from 6 bits to 4 bits. The first and last bit of each box provides the row index, and the remaining bits provide the column index. These indices are used to look-up values in a substitution box. A substitution box has 4 rows, 16 columns, and contains numbers from 0 to 15.
6. The result is transposed in accordance with the following rule:

```
// The array elements denote the bit numbers
int permutation_table[32] = {
    16,7,20,21,29,12,28,17,
    1,15,23,26,5,18,31,10,
    2,8,24,14,32,27,3,9,
    19,13,30,6,22,11,4,25
};
```

7. XOR the left half with the result from the above step. Store this in the right plain text.
8. Store the initial right plain text in the left plain text.
9. These halves are inputs for the next round. Remember that there are different keys for each round.
10. After the 16 rounds of encryption, swap the left plain text and the right plain text.
11. Finally, apply the inverse permutation (inverse of the initial permutation), and the ciphertext will be generated.

Steps for decryption:

The order of the 16 48-bit keys is reversed such that key 16 becomes key 1, and so on. Then, the steps for encryption are applied to the ciphertext.

Diffie Hellman Key Exchange Algorithm

Whitefield Diffie and Martin Hellman develop Diffie Hellman key exchange Algorithms in 1976 to overcome the problem of key agreement and exchange. It enables the two parties who want to communicate with each other to agree on a symmetric key, a key that can be used for encrypting and decryption; note that Diffie Hellman key exchange algorithm can be used for only key exchange, not for encryption and decryption process. The algorithm is based on mathematical principles.

The algorithm is based on Elliptic Curve Cryptography, a method of doing public-key cryptography based on the algebra structure of elliptic curves over finite fields. The DH also uses the trapdoor function, just like many other ways to do public-key cryptography. The simple idea of understanding to the DH Algorithm is the following.

1. The first party picks two prime numbers, g and p and tells them to the second party.

2. The second party then picks a secret number (let's call it a), and then it computes ga mod p and sends the result back to the first party; let's call the result A. Keep in mind that the secret number is not sent to anyone, only the result is.

3. Then the first party does the same; it selects a secret number b and calculates the result B similor to the

4. step 2. Then, this result is sent to the second party.

5. The second party takes the received number B and calculates B^a mod p

6. The first party takes the received number A and calculates A^b mod p

This is where it gets interesting; the answer in step 5 is the same as the answer in step 4. This means both parties will get the same answer no matter the order of exponentiation.

$(g^a \bmod p)^b \bmod p = g^{ab} \bmod p$

$(g^b \bmod p)^a \bmod p = g^{ba} \bmod p$

The number we came within steps 4 and 5 will be taken as the shared secret key. This key can be used to do any encryption of data that will be transmitted, such as blowfish, AES, etc

Diffie Hellman Algorithm

1. key $=(Y_A)^{X_B} \bmod q$ -> this is the same as calculated by B

2. Global Public Elements

- q: q is a prime number
- a: a < q and α is the primitive root of q

3. Key generation for user A

- Select a Private key X_A Here, X_A <q

Now, Calculation of Public key Y_A $Y_A = a^{X_A} \bmod q$

4. Key generation for user B

- Select a Private key X_B Here, X_B <q
- Now, Calculation of Public key Y_B $Y_B = a^{X_b} \bmod q$

5. Calculation of Secret Key by A

- key $=(Y_B)^{X_A} \bmod q$

6. Calculation of Secret Key by B

- key $=(Y_A)^{X_B} \bmod q$

Example

1. Alice and Bob both use public numbers P = 23, G = 5

2. Alice selected private key a = 4, and Bob selected b = 3 as the private key

3. Both Alice and bob now calculate the value of x and y as follows:

- Alice: $x = (5^4 \bmod 23) = 4$
- Bob: $y = (5^3 \bmod 23) = 10$

4. Now, both Alice and Bob exchange public numbers with each other.

5. Alice and Bob now calculate the symmetric keys

- Alice: $k_a = y^a \bmod p = 10^4 \bmod 23 = 18$
- Bob: $k_b = x^b \bmod p = 4^3 \bmod 23 = 18$

6. 18 is the shared secret key.

Uses of Diffie Hellman Algorithm

Aside from using the algorithm for generating public keys, there are some other places where DH Algorithm can be used:

Encryption: The Diffie Hellman key exchange algorithm can be used to encrypt; one of the first schemes to do is ElGamal encryption. One modern example of it is called Integrated Encryption Scheme, which provides security against chosen plain text and chosen clipboard attacks.

Password Authenticated Agreement: When two parties share a password, a password-authenticated key agreement can be used to prevent the Man in the middle attack. This key Agreement can be in the form of Diffie-Hellman. Secure Remote Password Protocol is a good example that is based on this technique.

Forward Secrecy: Forward secrecy-based protocols can generate new key pairs for each new session, and they can automatically discard them when the session is finished. In these forward Secrecy protocols, more often than not, the Diffie Hellman key exchange is used.

Advantages of the Diffie Hellman Algorithm

- The sender and receiver don't need any prior knowledge of each other.
- Once the keys are exchanged, the communication of data can be done through an insecure channel.
- The sharing of the secret key is safe.

Disadvantages of the Diffie Hellman Algorithm

- The algorithm can not be sued for any asymmetric key exchange.
- Similarly, it can not be used for signing digital signatures.
- Since it doesn't authenticate any party in the transmission, the Diffie Hellman key exchange is susceptible to a man-in-the-middle attack.