**Experiment No. 01**

**Aim:**

Write a Java/C/C++/Python program that contains a string (char pointer) with a value \HelloWorld’. The program should AND and XOR each character in this string with127 and display the result.

**Theory:**

The AND gate is an electronic circuit that gives a high output(1) only if all the inputs are high. A dot(.) is used to show the AND operation i.e. A.B this dot is sometimes omitted.

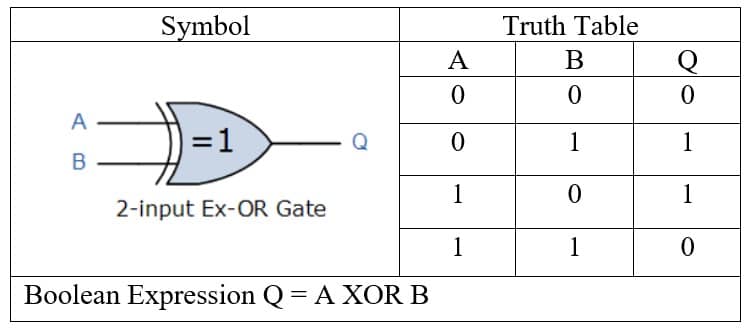
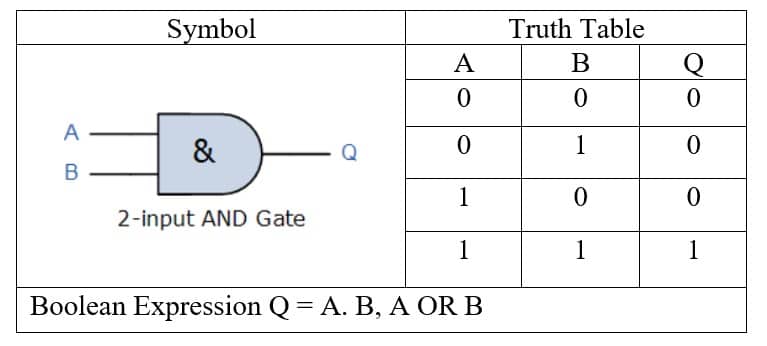


Fig: XOR Gate

Fig: AND Gate

**Algorithm:**

1. Start

2. Take the input ‘hello world’ which is assigned to variable ‘str’

3. Perform AND operation between the string and 127.

4. Then print the result

5. Stop.

**Conclusion:**

Thus we have studied XOR and AND gate operation on string to identify the encryption and decryption process.

**Experiment No. 02**

**Aim:**

Write a Java/C/C++/Python program to perform encryption and decryption using the method of Transposition technique.

**Theory:**

This is a C++ Program to implement transposition technique. In cryptography, a transposition cipher is methods of encryption by which the positions held by units of plaintext (which are commonly characters or groups of characters) are shifted according to a regular system, so that the ciphertext constitutes a permutation of the plaintext.

That is, the order of the units is changed (the plaintext is reordered). Mathematically a bijective function is used on the characters’ positions to encrypt and an inverse function to decrypt.

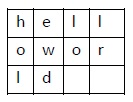
Transposition Cipher is a cryptographic algorithm where the order of alphabets in the plaintext is rearranged to form a cipher text. In this process, the actual plain text alphabets are not included.

**Example**

A simple example for a transposition cipher is columnar transposition cipher where each character in the plain text is written horizontally with specified alphabet width. The cipher is written vertically, which creates an entirely different cipher text.

Consider the plain text hello world, and let us apply the simple columnar transposition technique as shown below

**Columnar Transposition**



The plain text characters are placed horizontally and the cipher text is created with vertical format as : holewdlo lr. Now, the receiver has to use the same table to decrypt the cipher text to plain text.

The cipher text and the mentioned key are the two values taken as input parameters for decoding or decrypting the cipher text in reverse technique by placing characters in a column format and reading them in a horizontal manner.

**Conclusion:**

Thus we have studied C program to perform encryption and decryption using the method of Transposition technique.

**Experiment No. 03**

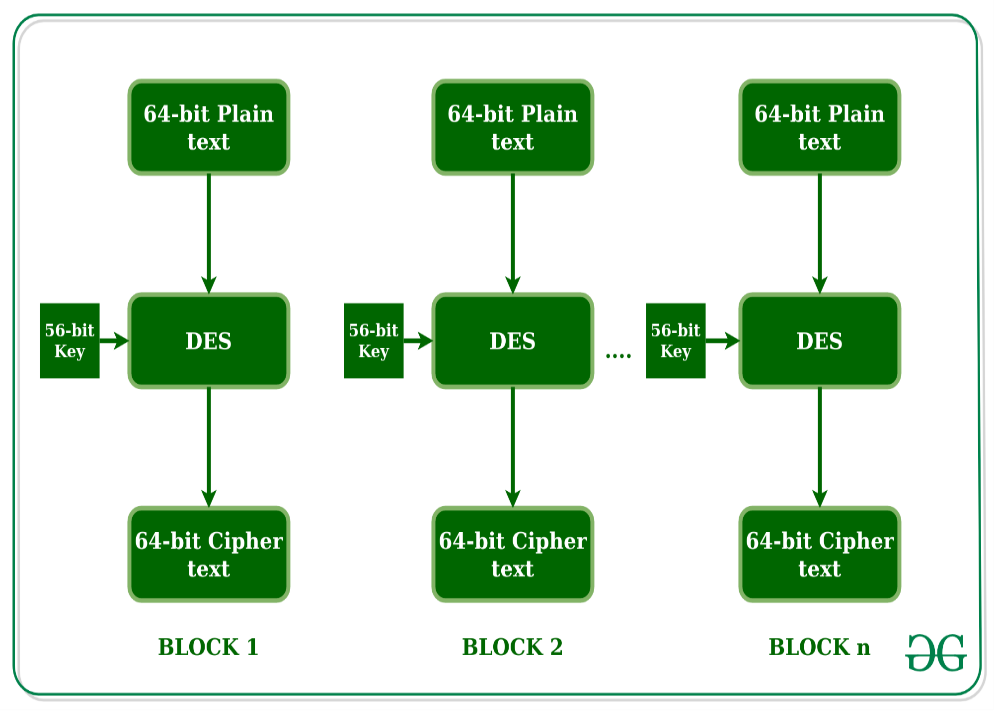
**Aim:**

Write a Java/C/C++/Python program to implement DES algorithm.

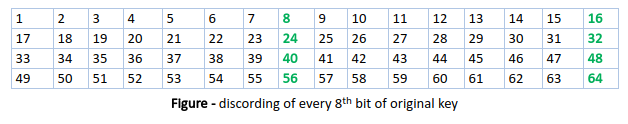
**Theory:**

Data encryption standard (DES) has been found vulnerable against very powerful attacks and therefore, the popularity of DES has been found slightly on the decline.

DES is a block cipher and encrypts data in blocks of size of 64 bits each, which means 64 bits of plain text goes as the input to DES, which produces 64 bits of ciphertext. The same algorithm and key are used for encryption and decryption, with minor differences. The key length is 56 bits. The basic idea is shown in the figure.



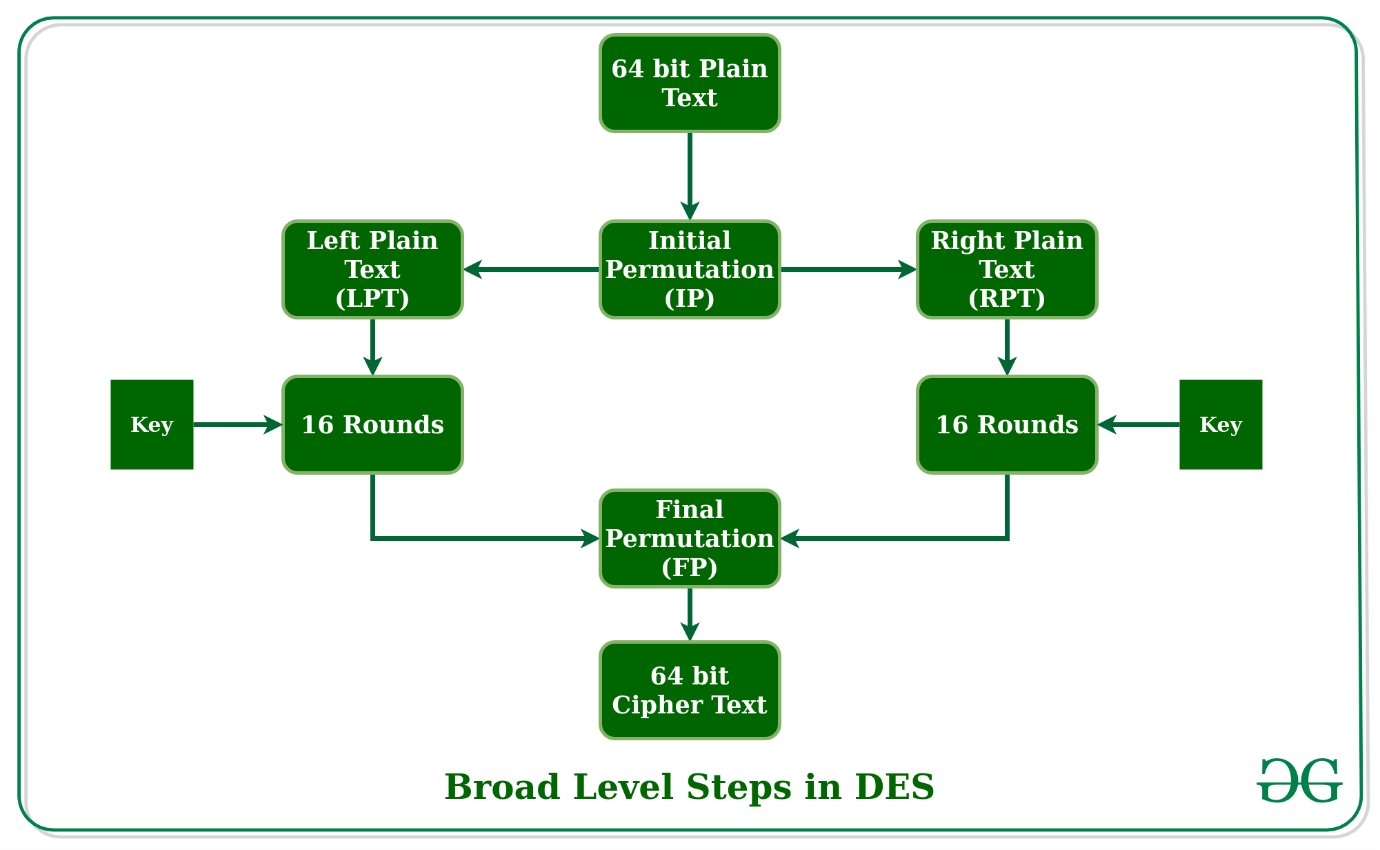
We have mentioned that DES uses a 56-bit key. Actually, the initial key consists of 64 bits. However, before the DES process even starts, every 8th bit of the key is discarded to produce a 56-bit key. That is bit positions 8, 16, 24, 32, 40, 48, 56, and 64 are discarded.



Thus, the discarding of every 8th bit of the key produces a 56-bit key from the original 64-bit key.

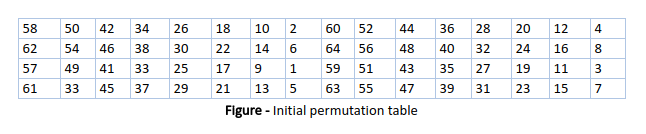
DES is based on the two fundamental attributes of cryptography: substitution (also called confusion) and transposition (also called diffusion). DES consists of 16 steps, each of which is called a round. Each round performs the steps of substitution and transposition. Let us now discuss the broad-level steps in DES.

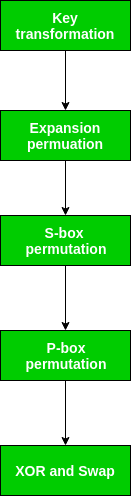
1. In the first step, the 64-bit plain text block is handed over to an initial Permutation (IP) function.
2. The initial permutation is performed on plain text.
3. Next, the initial permutation (IP) produces two halves of the permuted block; says Left Plain Text (LPT) and Right Plain Text (RPT).
4. Now each LPT and RPT go through 16 rounds of the encryption process.
5. In the end, LPT and RPT are rejoined and a Final Permutation (FP) is performed on the combined block
6. The result of this process produces 64-bit ciphertext.



**Initial Permutation (IP)**

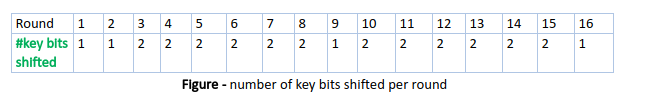
As we have noted, the initial permutation (IP) happens only once and it appens before the first round. It suggests how the transposition in IP should proceed, as shown in the figure.   
For example, it says that the IP replaces the first bit of the original plain text block with the 58th bit of the original plain text, the second bit with the 50th bit of the original plain text block, and so on.  
This is nothing but jugglery of bit positions of the original plain text block. the same rule applies to all the other bit positions shown in the figure.

 As we have noted after IP is done, the resulting 64-bit permuted text block is divided into two half blocks. Each half-block consists of 32 bits, and each of the 16 rounds, in turn, consists of the broad level steps outlined in the figure.

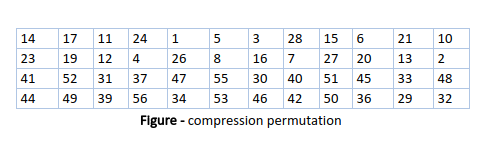


**Step-1: Key transformation –**

We have noted initial 64-bit key is transformed into a 56-bit key by discarding every 8th bit of the initial key. Thus, for each a 56-bit key is available. From this 56-bit key, a different 48-bit Sub Key is generated during each round using a process called key transformation. For this, the 56-bit key is divided into two halves, each of 28 bits. These halves are circularly shifted left by one or two positions, depending on the round.  
For example, if the round numbers 1, 2, 9, or 16 the shift is done by only position for other rounds, the circular shift is done by two positions. The number of key bits shifted per round is shown in the figure.



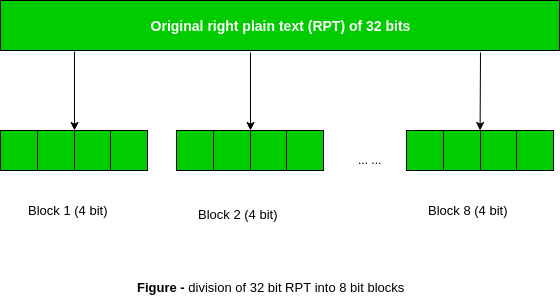
After an appropriate shift, 48 of the 56 bits are selected. for selecting 48 of the 56 bits the table is shown in the figure given below. For instance, after the shift, bit number 14 moves on the first position, bit number 17 moves on the second position, and so on. If we observe the table carefully, we will realize that it contains only 48-bit positions. Bit number 18 is discarded (we will not find it in the table), like 7 others, to reduce a 56-bit key to a 48-bit key. Since the key transformation process involves permutation as well as a selection of a 48-bit subset of the original 56-bit key it is called Compression Permutation.



Because of this compression permutation technique, a different subset of key bits is used in each round. That makes DES not easy to crack.

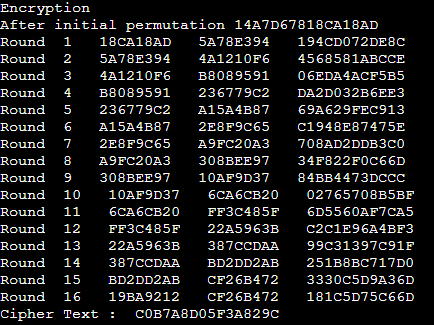
**Step-2: Expansion Permutation –**

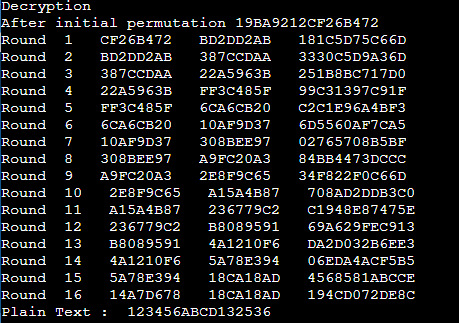
Recall that after initial permutation, we had two 32-bit plain text areas called Left Plain Text(LPT) and Right Plain Text(RPT). During the expansion permutation, the RPT is expanded from 32 bits to 48 bits. Bits are permuted as well hence called expansion permutation. This happens as the 32-bit RPT is divided into 8 blocks, with each block consisting of 4 bits. Then, each 4-bit block of the previous step is then expanded to a corresponding 6-bit block, i.e., per 4-bit block, 2 more bits are added.



This process results in expansion as well as a permutation of the input bit while creating output. The key transformation process compresses the 56-bit key to 48 bits. Then the expansion permutation process expands the 32-bit RPT to 48-bits. Now the 48-bit key is XOR with 48-bit RPT and the resulting output is given to the next step, which is the S-Box substitution.

**Output:**





**Conclusion:**

Thus we have studied encryption and decryption using DES algorithm

**Experiment No. 4**

**Aim:**

Write a Java/C/C++/Python program to implement AES Algorithm

**Theory:**

Advanced Encryption Standard (AES) is a specification for the encryption of electronic data established by the U.S National Institute of Standards and Technology (NIST) in 2001. AES is widely used today as it is a much stronger than DES and triple DES despite being harder to implement.

Points to remember

* AES is a block cipher.
* The key size can be 128/192/256 bits.
* Encrypts data in blocks of 128 bits each.

That means it takes 128 bits as input and outputs 128 bits of encrypted cipher text as output. AES relies on substitution-permutation network principle which means it is performed using a series of linked operations which involves replacing and shuffling of the input data.

**Working of the cipher:**

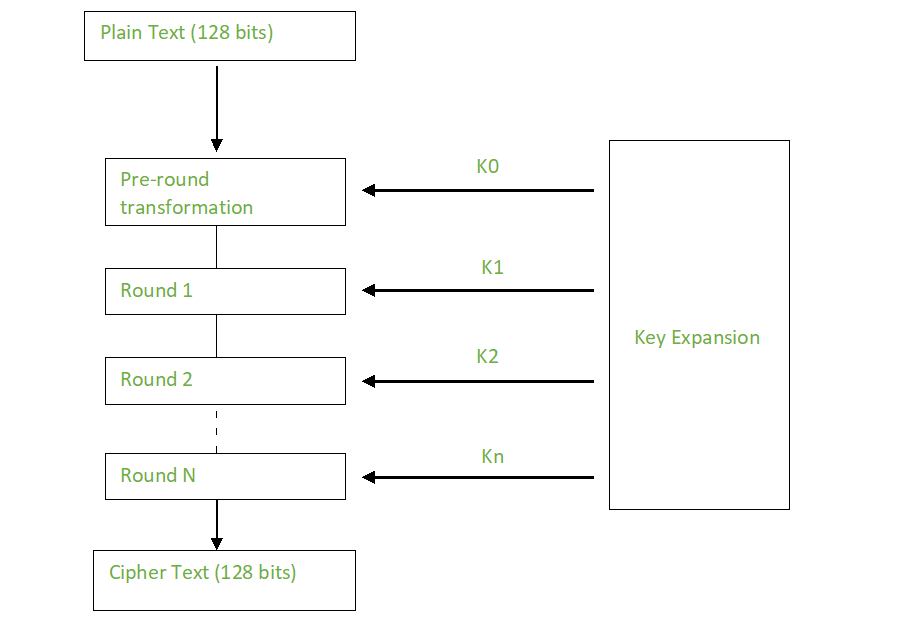
AES performs operations on bytes of data rather than in bits. Since the block size is 128 bits, the cipher processes 128 bits (or 16 bytes) of the input data at a time.

The number of rounds depends on the key length as follows :

* 128 bit key – 10 rounds
* 192 bit key – 12 rounds
* 256 bit key – 14 rounds

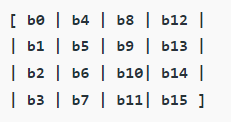
**Creation of Round keys :**

A Key Schedule algorithm is used to calculate all the round keys from the key. So the initial key is used to create many different round keys which will be used in the corresponding round of the encryption.



**Encryption :**

AES considers each block as a 16 byte (4 byte x 4 byte = 128 ) grid in a column major arrangement.



Each round comprises of 4 steps :

* SubBytes
* ShiftRows
* MixColumns
* Add Round Key

The last round doesn’t have the MixColumns round.

The SubBytes does the substitution and ShiftRows and MixColumns performs the permutation in the algorithm.

**SubBytes :**

This step implements the substitution.

In this step each byte is substituted by another byte.(Its performed using a lookup table also called the S-box.This substitution is done in a way that a byte is never substituted by itself and also not substituted by another byte which is a compliment of the current byte.The result of this step is a 16 byte (4 x 4 ) matrix like before.

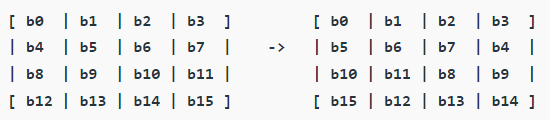
The next two steps implement the permutation.

**ShiftRows :**

This step is just as it sounds. Each row is shifted a particular number of times.

* The first row is not shifted
* The second row is shifted once to the left.
* The third row is shifted twice to the left.
* The fourth row is shifted thrice to the left.

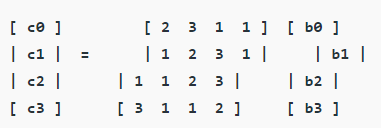
(A left circular shift is performed.)



**MixColumns :**

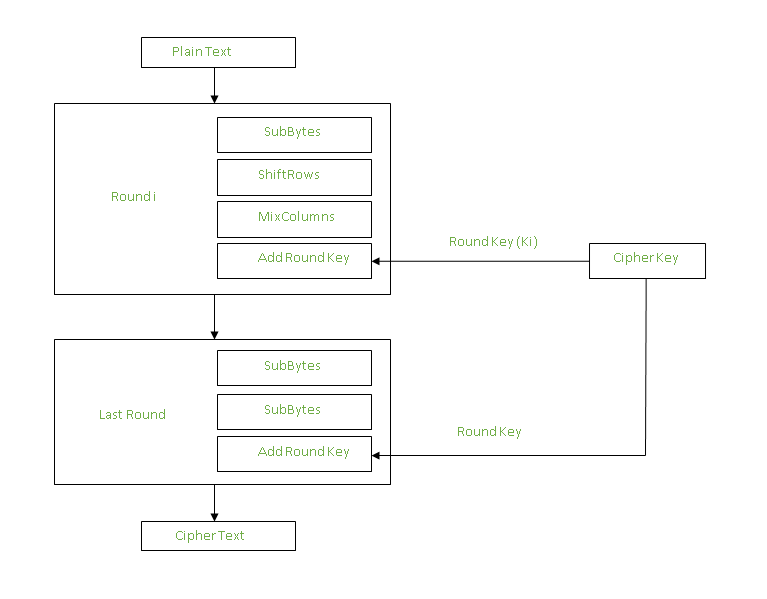
This step is basically a matrix multiplication. Each column is multiplied with a specific matrix and thus the position of each byte in the column is changed as a result.

**This step is skipped in the last round.**



**Add Round Keys :**

Now the resultant output of the previous stage is XOR-ed with the corresponding round key. Here, the 16 bytes is not considered as a grid but just as 128 bits of data.



After all these rounds 128 bits of encrypted data is given back as output. This process is repeated until all the data to be encrypted undergoes this process.

**Decryption :**

The stages in the rounds can be easily undone as these stages have an opposite to it which when performed reverts the changes.Each 128 blocks goes through the 10,12 or 14 rounds depending on the key size.

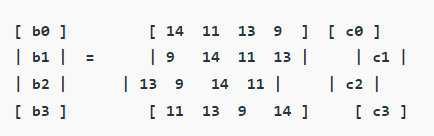
The stages of each round in decryption is as follows :

* Add round key
* Inverse MixColumns
* ShiftRows
* Inverse SubByte

The decryption process is the encryption process done in reverse so i will explain the steps with notable differences.

**Inverse MixColumns :**

This step is similar to the MixColumns step in encryption, but differs in the matrix used to carry out the operation.



**Inverse SubBytes :**

Inverse S-box is used as a lookup table and using which the bytes are substituted during decryption.

**Conclusion:**

Thus we have studied encryption and decryption using AES algorithm

**Experiment no. 05**

**Aim:**

Write a Java/C/C++/Python program to implement RSA algorithm

**Theory:**

RSA (Rivest–Shamir–Adleman) algorithm is asymmetric cryptography algorithm. The [acronym](https://en.wikipedia.org/wiki/Acronym) "RSA" comes from the surnames of [Ron Rivest](https://en.wikipedia.org/wiki/Ron_Rivest), [Adi Shamir](https://en.wikipedia.org/wiki/Adi_Shamir" \o "Adi Shamir) and [Leonard Adleman](https://en.wikipedia.org/wiki/Leonard_Adleman), who publicly described the algorithm in 1977.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 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 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.

**Let us learn the mechanism behind RSA algorithm :**

**Generating Public Key :**

1. Select two prime no's. Suppose **P = 53 and Q = 59**.
2. Now First part of the Public key : **n = P\*Q = 3127**.
3. We also need a small exponent say **e** :
4. But e Must be
   1. An integer.
   2. Not be a factor of n.
   3. **1 < e <** [Φ(n)](https://www.geeksforgeeks.org/eulers-totient-function/) [Φ(n) is discussed below],
   4. Let us now consider it to be equal to 3.
5. Our Public Key is made of n and e
   1. **>> Generating Private Key :**
6. We need to calculate Φ(n) :
7. Such that **Φ(n) = (P-1)(Q-1)**
8. so, Φ(n) = 3016
9. Now calculate Private Key, **d** :
10. **d = (k\*Φ(n) + 1) / e** for some integer k
11. For k = 2, value of d is 2011.
    1. Now we are ready with our – Public Key ( n = 3127 and e = 3) and Private Key(d = 2011)
    2. Now we will encrypt **“HI”** :
12. Convert letters to numbers : H = 8 and I = 9
13. Thus **Encrypted Data c = 89e mod n**.
14. Thus our Encrypted Data comes out to be 1394
15. Now we will decrypt **1394** :
16. **Decrypted Data = cd mod n**.
17. Thus our Encrypted Data comes out to be 89
18. **8 = H and I = 9 i.e. "HI".**

**Output:**



**Conclusion:**

Thus we have studied RSA (Rivest–Shamir–Adleman) algorithm, a asymmetric cryptography algorithm.

**Experiment No. 06**

**Aim:**

Implement the Diffie-Hellman Key Exchange mechanism using HTML and JavaScript. Consider the end user as one of the parties (Alice) and the JavaScript application as other party (bob).

**Theory:**

The Diffie-Hellman algorithm is being used to establish a shared secret that can be used for secret communications while exchanging data over a public network using the elliptic curve to generate points and get the secret key using the parameters.

For the sake of simplicity and practical implementation of the algorithm, we will consider only 4 variables, one prime P and G (a primitive root of P) and two private values a and b.

P and G are both publicly available numbers. Users (say Alice and Bob) pick private values a and b and they generate a key and exchange it publicly. The opposite person receives the key and that generates a secret key, after which they have the same secret key to encrypt.

**Step by Step Explanation**

**Alice Bob**

Public Keys available = P, G Public Keys available = P, G

Private Key Selected = a Private Key Selected = b

Key generated = Key generated =

x = G^a mod P y = G^b mod P

Exchange of generated keys takes place

Key received = y key received = x

Generated Secret Key = Generated Secret Key =

ka = y^a mod P kb = x^b mod P

Algebraically, it can be shown that

ka = kb

Users now have a symmetric secret key to encrypt

**Example:**

Step 1: Alice and Bob get public numbers P = 23, G = 9

Step 2: Alice selected a private key a = 4 and

Bob selected a private key b = 3

Step 3: Alice and Bob compute public values

Alice: x =(9^4 mod 23) = (6561 mod 23) = 6

Bob: y = (9^3 mod 23) = (729 mod 23) = 16

Step 4: Alice and Bob exchange public numbers

Step 5: Alice receives public key y =16 and

Bob receives public key x = 6

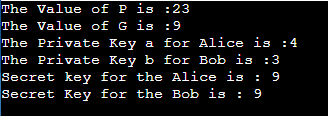
Step 6: Alice and Bob compute symmetric keys

Alice: ka = y^a mod p = 65536 mod 23 = 9

Bob: kb = x^b mod p = 216 mod 23 = 9

Step 7: 9 is the shared secret.

**Output:**



**Conclusion:**

Thus we have studied how to exchange keys using Diffie-Hellman key exchange.