

Assignment No : 1	Date:
Title: Write a Java/C/C++/Python program that contains a string (char pointer) with a value \Hello World'. The program should AND or and XOR each character in this string with 127 and display the result.	
Sign:	Remark:

AIM: Write a C program that contains a string (char pointer) with a value \Hello World'. The program should AND or and XOR each character in this string with 127 and display the result.

HARDWARE AND SOFTWARE REQUIREMENT:

1. Intel based Desktop PC: - RAM of 512 MB
2. Turbo C / Borland C

THEORY:

(a) String

The string is the one-dimensional array of characters terminated by a null ('\0'). Each and every character in the array consumes one byte of memory, and the last character must always be 0. The termination character ('\0') is used to identify where the string ends. In C language string declaration can be done in two ways

1. By char array
2. By string literal

Let's see the example of declaring **string by char array** in C language.

1. **char** ch[17]={ 'o', 'n', 'l', 'i', 'n', 'e', 's', 'm', 'a', 'r', 't', 't', 'r', 'a',
'i', 'n', 'e', 'r', '\0'};

As we know, array index starts from 0, so it will be represented as in the figure given below.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
o	n	l	i	n	e	s	m	a	r	t	t	r	a	i	n	e	r	\0

While declaring string, size is not mandatory. So we can write the above code as given below:

char ch[]={ 'o', 'n', 'l', 'i', 'n', 'e', 's', 'm', 'a', 'r', 't', 't', 'r', 'a',
'i', 'n', 'e', 'r', '\0'};

We can also define the **string by the string literal** in C language. For example:

2. **char** str[]="onlinesmarttrainer";

In such case, '\0' will be appended at the end of the string by the compiler.

(b) AND Operation

There are two inputs and one output in binary **AND** operation. The inputs and result to a binary **AND** operation can only be **0** or **1**. The binary **AND** operation will always produce a **1** output if both inputs are **1** and will produce a **0** output if both inputs are **0**. For two different inputs, the output will be **0**.

AND Truth Table

Input		Output
X	Y	
0	0	0
0	1	0
1	0	0
1	1	1

(c) XOR Operation

There are two inputs and one output in binary **XOR** (exclusive **OR**) operation.

It is similar to **ADD** operation which takes two inputs and produces one result i.e. one output.

The inputs and result to a binary **XOR** operation can only be **0** or **1**. The binary **XOR** operation will always produce a **1** output if either of its inputs is **1** and will produce a **0** output if both of its inputs are **0** or **1**.

XOR Truth Table

Input		Output
X	Y	
0	0	0
0	1	1
1	0	1
1	1	0

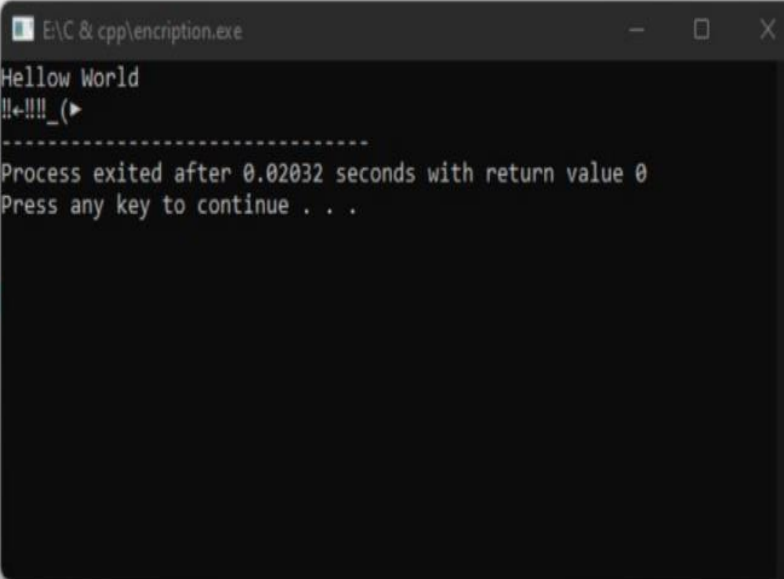
SOURCE CODE & OUTPUT:

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <stdlib.h>

int main()
{
    char str[]="Hellow World";
    char str1[11];
    char str2[11];
    int i,len;
    len = strlen(str);

    for(i=0;i<len;i++)
    {
        str1[i]=str[i] & 127;
        printf("%c",str1[i]);
    }
    printf("\n");

    for(i=0;i<len;i++)
    {
        str1[i]=str[i]^127;
        printf("%c",str1[i]);
    }
}
```



```
E:\C & cpp\encrption.exe
Hellow World
!!<!!!!_>
-----
Process exited after 0.02032 seconds with return value 0
Press any key to continue . . .
```

Assignment No : 2	Date:
Title: Write a Java/C/C++/Python program to perform encryption and decryption using the method of Transposition technique.	
Sign:	Remark:

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

Theory:

In cryptography, a transposition cipher is a method 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.

1. Encryption

In a transposition cipher, the order of the alphabets is re-arranged to obtain the ciphertext.

- a) The message is written out in rows of a fixed length, and then read out again column by column, and the columns are chosen in some scrambled order.
- b) Width of the rows and the permutation of the columns are usually defined by a keyword.
- c) For example, the word HACK is of length 4 (so the rows are of length 4), and the permutation is defined by the alphabetical order of the letters in the keyword. In this case, the order would be “3 1 2 4”.
- d) Any spare spaces are filled with nulls or left blank or placed by a character (Example: _).
- e) Finally, the message is read off in columns, in the order specified by the keyword.

Given Text = College Life

Keyword = HACK

Order Of Alphabet=3124

H	A	C	K
3	1	2	4
C	o	l	l
e	g	e	—
L	i	f	e

Encrypted Text= ogilefCeLI_e

2. Decryption

- To decipher it, the recipient has to work out the column lengths by dividing the message length by the key length.
- Then, write the message out in columns again, then re-order the columns by re-forming the key word.

Question

The following message was encrypted with a columnar transposition cipher using a full rectangular array and keyword mathematics. Decrypt it.

RIUGS IPNCT MSPAL AUNCY SOOCH UEYSA RTE

Assignment No :3	Date:
Title: Implementation of DES.	
Sign:	Remark:

Title: Implementation of DES

Problem Definition: Write a Java / Python program for performs the various operations on DES

Prerequisite:

- Basic concepts of Java /Python/C++

Tools/Framework/Language Used: Java / Python/C++

Learning Objectives: Understand the implementation of DES key generation passing keys, encryption and decryption.

Outcomes: After completion of this assignment students will understand how Encrypt and Decrepit work.

Theory:

Concepts:

DES, developed by Professor Edward Schaefer of Santa Clara University [SCHA96], is an educational rather than a secure encryption algorithm. It has similar properties and structure to DES with much smaller parameters. The reader might find it useful to work through an example byhand while following the discussion in this Appendix.

OVERVIEW

The S-DES encryption algorithm takes an 8-bit block of plaintext (example: 10111101) and a 10-bitkey as input and produces an 8-bit block of ciphertext as output. The S-DES decryption algorithm takes an 8-bit block of ciphertext and the same 10-bit key used to produce that ciphertext as input and produces the original 8-bit block of plaintext.

The encryption algorithm involves five functions: an initial permutation (IP); a complex function labeled f_K , which involves both permutation and substitution operations and depends on a key input; a simple permutation function that switches (SW) the two halves of the data; the function f_K again; and finally a permutation function that is the inverse of the initial permutation (IP^{-1}). As was mentioned in Chapter 2, the use of multiple stages of permutation and substitution results in a more complex algorithm, which increases the difficulty of cryptanalysis.

The function f_K takes as input not only the data passing through the encryption algorithm, but also an 8-bit key. The algorithm could have been designed to work with a 16-bit key, consisting of two 8-bit subkeys, one used for each occurrence of f_K . Alternatively, a single 8-bit key could have been used, with the same key used twice in the algorithm. A compromise is to use a 10-bit key from which two 8-bit subkeys are generated, as depicted in Figure G.1. In this case, the key is first subjected to a permutation (P10). Then a shift operation is performed. The output of the shift operation then passes through a permutation function that produces an 8-bit output (P8) for the first subkey (K_1). The output of the shift operation also feeds into another shift and another instance of P8 to produce the second subkey (K_2).

We can concisely express the encryption algorithm as a composition¹ of functions:

$$IP^{-1} \circ f_{K_2} \circ SW \circ f_{K_1} \circ IP$$

which can also be written as:

$$\text{ciphertext} = IP^{-1} \left(f_{K_2} \left(SW \left(f_{K_1} \left(IP(\text{plaintext}) \right) \right) \right) \right)$$

where

$$K_1 = P8 \left(\text{Shift} \left(P10(\text{key}) \right) \right)$$

$$K_2 = P8 \left(\text{Shift} \left(\text{Shift} \left(P10(\text{key}) \right) \right) \right)$$

Decryption is also shown in Figure G.1 and is essentially the reverse of encryption:

$$\text{plaintext} = IP^{-1} \left(f_{K_1} \left(SW \left(f_{K_2} \left(IP(\text{ciphertext}) \right) \right) \right) \right)$$

We now examine the elements of S-DES in more detail.

DES KEY GENERATION

S-DES depends on the use of a 10-bit key shared between sender and receiver. From this key, two 8-bit subkeys are produced for use in particular stages of the encryption and decryption algorithm. Figure G.2 depicts the stages followed to produce the subkeys.

First, permute the key in the following fashion. Let the 10-bit key be designated as $(k_1, k_2, k_3, k_4, k_5, k_6, k_7, k_8, k_9, k_{10})$. Then the permutation P10 is defined as:

$$P10(k_1, k_2, k_3, k_4, k_5, k_6, k_7, k_8, k_9, k_{10}) = (k_3, k_5, k_2, k_7, k_4, k_{10}, k_1, k_9, k_8, k_6)$$

4. **Definition:** If f and g are two functions, then the function F with the equation $y = F(x) = g[f(x)]$ is called the **composition** of f and g and is denoted as $F = g \circ f$.

P10 can be concisely defined by the display:

P10									
3	5	2	7	4	10	1	9	8	6

This table is read from left to right; each position in the table gives the identity of the input bit that produces the output bit in that position. So the first output bit is bit 3 of the input; the second output bit is bit 5 of the input, and so on. For example, the key (1010000010) is permuted to (1000001100). Next, perform a circular left shift (LS-1), or rotation, separately on the first five bits and the second five bits. In our example, the result is (00001 11000).

Next we apply P8, which picks out and permutes 8 of the 10 bits according to the following rule:

P8							
6	3	7	4	8	5	10	9

The result is subkey 1 (K_1). In our example, this yields (10100100)

We then go back to the pair of 5-bit strings produced by the two LS-1 functions and perform a circular left shift of 2 bit positions on each string. In our example, the value (00001 11000) becomes (00100 00011). Finally, P8 is applied again to produce K_2 . In our example, the result is (01000011).

DES ENCRYPTION

Figure G.3 shows the S-DES encryption algorithm in greater detail. As was mentioned, encryption involves the sequential application of five functions. We examine each of these.

Initial and Final Permutations

The input to the algorithm is an 8-bit block of plaintext, which we first permute using the IP function:

	IP						
2	6	3	1	4	8	5	7

This retains all 8 bits of the plaintext but mixes them up. At the end of the algorithm, the inverse permutation is used:

	IP ⁻¹						
4	1	3	5	7	2	8	6

It is easy to show by example that the second permutation is indeed the reverse of the first; that is, $IP^{-1}(IP(X)) = X$.

The Function f_K

The most complex component of S-DES is the function f_K , which consists of a combination of permutation and substitution functions. The functions can be expressed as follows. Let L and R be the leftmost 4 bits and rightmost 4 bits of the 8-bit input to f_K , and let F be a mapping (not necessarily one to one) from 4-bit strings to 4-bit strings. Then we let

$$f_K(L, R) = (L \oplus F(R, SK), R)$$

where SK is a subkey and $!$ is the bit-by-bit exclusive-OR function. For example, suppose the output of the IP stage in Figure G.3 is (10111101) and $F(1101, SK) = (1110)$ for some key SK . Then $f_K(10111101) = (01011101)$ because $(1011) ! (1110) = (0101)$.

We now describe the mapping F . The input is a 4-bit number ($n_1 n_2 n_3 n_4$). The first operation is an expansion/permutation operation:

$$\begin{array}{ccccccc} & & & & \text{E/P} & & \\ & & & & & & \\ 4 & 1 & 2 & & 3 & 2 & 3 & 4 & 1 \end{array}$$

For what follows, it is clearer to depict the result in this fashion:

$$\begin{array}{c|cc|c} n_4 & n_1 & n_2 & n_3 \\ \hline n_2 & n_3 & n_4 & n_1 \end{array}$$

The 8-bit subkey $K_1 = (k_{11}, k_{12}, k_{13}, k_{14}, k_{15}, k_{16}, k_{17}, k_{18})$ is added to this value using exclusive-OR:

$$\begin{array}{c|cc|cc|c} n_4 ! k_{11} & n_1 ! k_{12} & n_2 ! k_{13} & n_3 ! k_{14} \\ \hline n_2 ! k_{15} & n_3 ! k_{16} & n_4 ! k_{17} & n_1 ! k_{18} \end{array}$$

Let us rename these 8 bits:

$$\begin{array}{c|cc|c} p_{0,0} & p_{0,1} & p_{0,2} & p_{0,3} \\ \hline 1,0 & 1,1 & 1,2 & 1,3 \end{array}$$

The first 4 bits (first row of the preceding matrix) are fed into the S-box S_0 to produce a 2-bit output, and the remaining 4 bits (second row) are fed into S_1 to produce another 2-bit output. These two boxes are defined as follows:

$$S0 = \begin{matrix} & 0 & 1 & 2 & 3 \\ \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \end{matrix} & \begin{bmatrix} 1 & 0 & 3 & 2 \\ 3 & 2 & 1 & 0 \\ 0 & 2 & 1 & 3 \\ 3 & 1 & 3 & 2 \end{bmatrix} \end{matrix} \quad S1 = \begin{matrix} & 0 & 1 & 2 & 3 \\ \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \end{matrix} & \begin{bmatrix} 0 & 1 & 2 & 3 \\ 2 & 0 & 1 & 3 \\ 3 & 0 & 1 & 0 \\ 2 & 1 & 0 & 3 \end{bmatrix} \end{matrix}$$

The S-boxes operate as follows. The first and fourth input bits are treated as a 2-bit number that specify a row of the S-box, and the second and third input bits specify a column of the S-box. The entry in that row and column, in base 2, is the 2-bit output. For example, if $(p_{0,0}p_{0,3}) =$

(0) and $(p_{0,1}p_{0,2}) = (10)$, then the output is from row 0, column 2 of S0, which is 3, or (11) in binary. Similarly, $(p_{1,0}p_{1,3})$ and $(p_{1,1}p_{1,2})$ are used to index into a row and column of S1 to produce an additional 2 bits.

Next, the 4 bits produced by S0 and S1 undergo a further permutation as follows:

P4			
2	4	3	1

The output of P4 is the output of the function F.

The Switch Function

The function f_K only alters the leftmost 4 bits of the input. The switch function (SW) interchanges the left and right 4 bits so that the second instance of f_K operates on a different 4 bits. In this second instance, the E/P, S0, S1, and P4 functions are the same. The key input is K_2 .

ANALYSIS OF DES

A brute-force attack on simplified DES is certainly feasible. With a 10-bit key, there are only $2^{10} = 1024$ possibilities. Given a ciphertext, an attacker can try each possibility and analyze the result to determine if it is reasonable plaintext.

What about cryptanalysis? Let us consider a known plaintext attack in which a single plaintext ($p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8$) and its ciphertext output ($c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8$) are known and the key ($k_1, k_2, k_3, k_4, k_5, k_6, k_7, k_8, k_9, k_{10}$) is unknown. Then each c_i is a polynomial function g_i of the p_j 's and k_j 's. We can therefore express the encryption algorithm as 8 nonlinear equations in 10 unknowns. There are a number of possible solutions, but each of these could be calculated and then analyzed. Each of the permutations and additions in the algorithm is a linear mapping. The nonlinearity comes from the S-boxes. It is useful to write down the equations for these boxes. For clarity, rename ($p_{0,0}, p_{0,1}, p_{0,2}, p_{0,3}$) = (a, b, c, d) and ($p_{1,0}, p_{1,1}, p_{1,2}, p_{1,3}$) = (w, x, y, z), and let the 4-bit output be (q, r, s, t). Then the operation of the S0 is defined by the following equations:

$$q = abcd + ab + ac + b + d$$

$$r = abcd + abd + ab + ac + ad + a + c + 1$$

where all additions are modulo 2. Similar equations define S1. Alternating linear maps with these nonlinear maps results in very complex polynomial expressions for the ciphertext bits, making cryptanalysis difficult. To visualize the scale of the problem, note that a polynomial equation in 10 unknowns in binary arithmetic can have 2^{10} possible terms. On average, we might therefore expect each of the 8 equations to have 2^9 terms. The interested reader might try to find these equations with a symbolic processor. Either the reader or the software will give up before much progress is made.

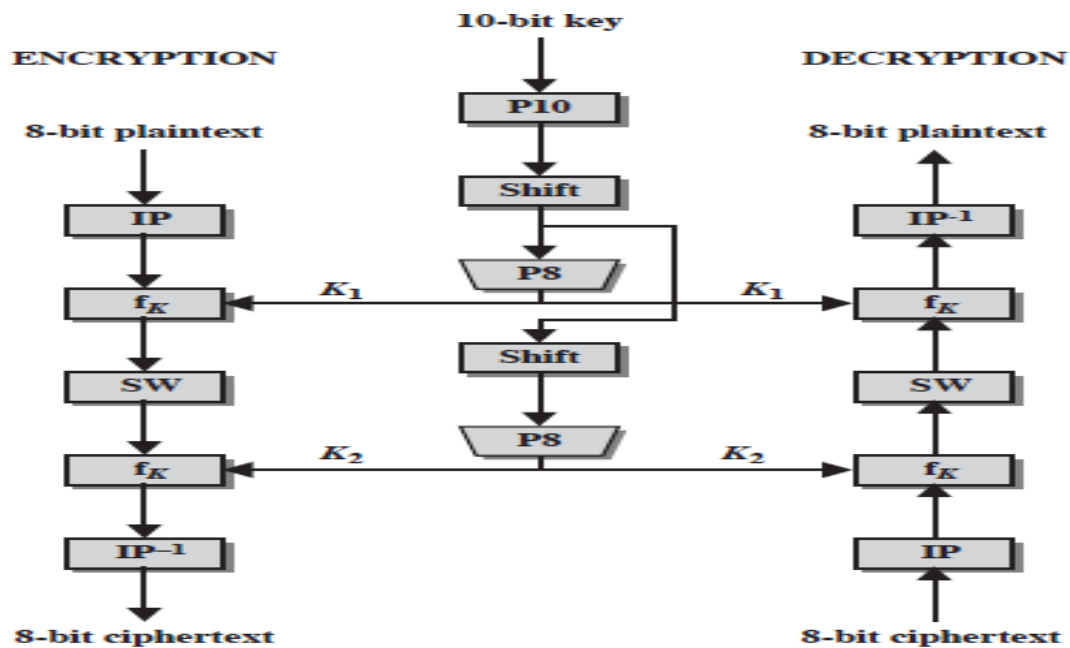


Figure G.1 Simplified DES Scheme

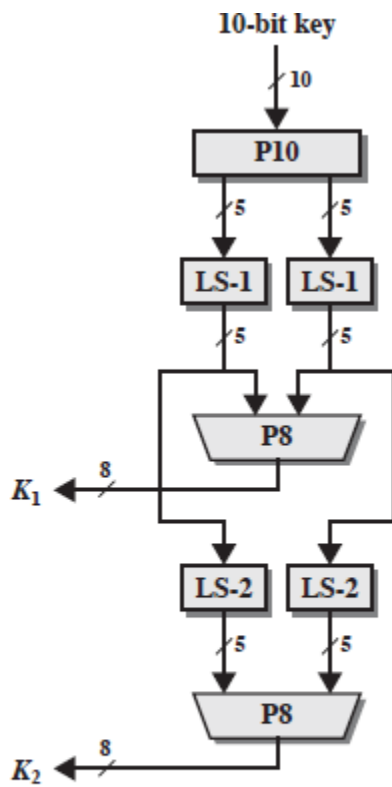


Figure G.2 Key Generation for Simplified DES

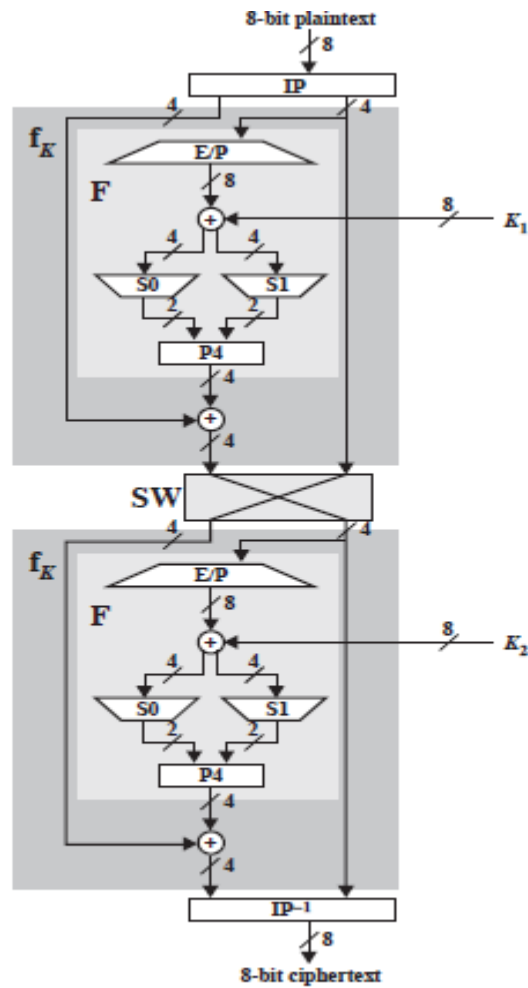


Figure G.3 Simplified DES Encryption Detail

Conclusion:

Thus we have implemented SDES and study the encryption and decryption process with key generation technique. DES encryption with two keys instead of one key already will increase the efficiency of cryptography.

Assignment No : 4	Date:
Title: Implementation of AES.	
Sign:	Remark:

Title: Implementation of AES

Problem Definition: Write a Java / Python program for performs the various operations on AES

Prerequisite:

- Basic concepts of Java /Python/C++

Tools/Framework/Language Used: Java / Python/C++

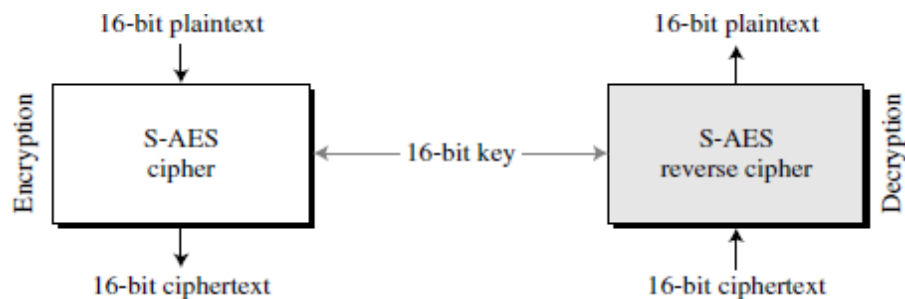
Learning Objectives: Understand the implementation of AES key generation passing keys, encryption and decryption.

Outcomes: After completion of this assignment students will understand how Encrypt and Decrepit work.

Theory:

Concepts:

AES is developed by Professor Edward Schaefer of Santa Clara University, is an educational tool designed to help students learn the structure of AES using smaller blocks and keys.



AES is a block cipher, as shown in Figure 1.

Rounds

S-AES is a non-Feistel cipher that encrypts and decrypts a data block of 16 bits. It uses one pre-round transformation and two rounds. The cipher key is also 16 bits.

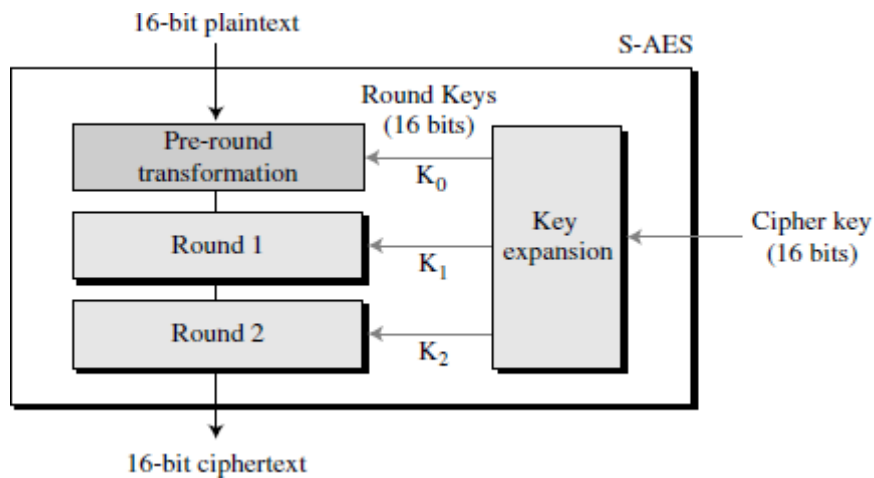
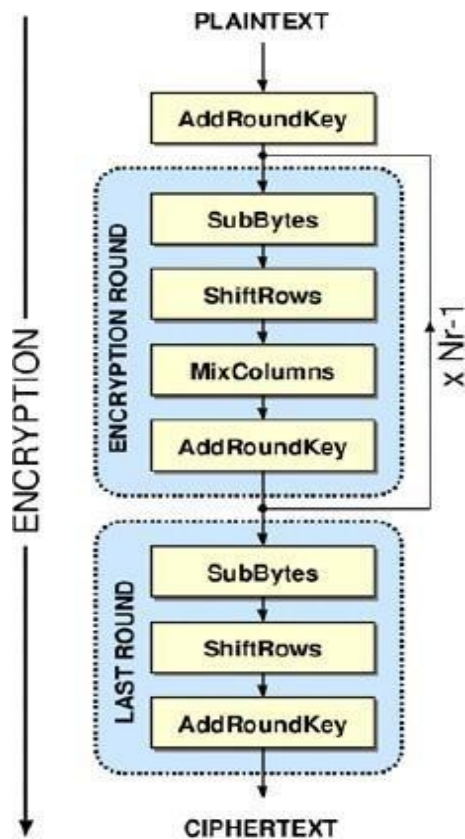


Figure P.2 Structure of AES

The above Fig. shows the general design for the encryption algorithm (called the cipher); the decryption algorithm (called the inverse cipher) is similar, but the round keys are applied in the reverse order. In Figure P.2, the round keys, which are created by the key-expansion algorithm, are always 16 bits, the same size as the plaintext or ciphertext block. In S-AES, there are three round keys, K_0 , K_1 , and K_2 .



Substitution

Substitution is done for each nibble (4-bit data unit). Only one table is used for transformations of every nibble, which means that if two nibbles are the same, the transformation is also the same. In this appendix, transformation is defined by a table lookup process.

SubNibbles

The first transformation, **SubNibbles**, is used at the encryption site. To substitute a nibble, we interpret the nibble as 4 bits. The left 2 bits define the row and the right 2 bits define the column of the substitution table. The hexadecimal digit at the junction of the row and the column is the new nibble.

In the SubNibbles transformation, the state is treated as a 2×2 matrix of nibbles. Transformation is done one nibble at a time. The contents of each nibble is changed, but the arrangement of the nibbles in the matrix remains the same. In the process, each nibble is transformed independently: There are four distinct nibble-to-nibble transformations.

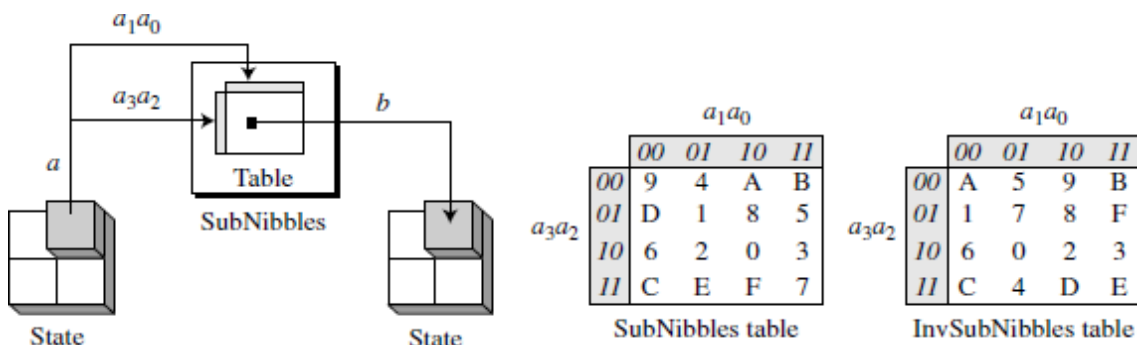


Figure P.7
SubNibbles transformations

Permutation

Another transformation found in a round is shifting, which permutes the nibbles. Shifting transformation in S-AES is done at the nibble level; the order of the bits in the nibble is not changed.

ShiftRows

In the encryption, the transformation is called *ShiftRows* and the shifting is to the left. The number of shifts depends on the row number (0, 1) of the state matrix. This means row 0 is not shifted at all and row 1 is shifted 1 nibble. Figure P.9 shows the shifting transformation. Note that the ShiftRows transformation operates one row at a time.

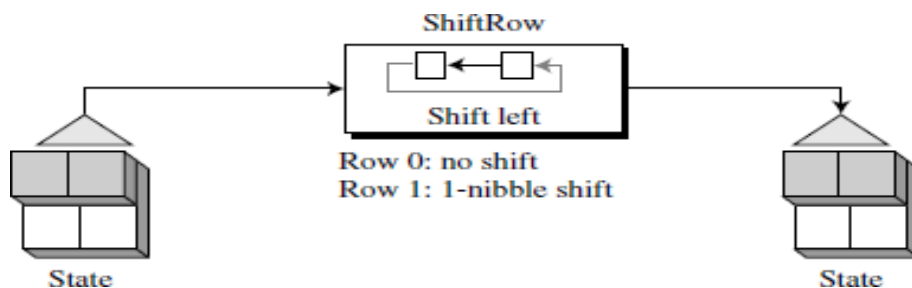


Figure P.9 *ShiftRows transformation*

MixColumns

The *MixColumns* transformation operates at the column level; it transforms each column of the state into a new column. The transformation is actually the matrix multiplication of a state column by a constant square matrix. The nibbles in the state column and constants matrix are interpreted as 4-bit words (or polynomials) with coefficients in GF(2). Multiplication of bytes is done in GF(24) with modulus (x^4+x+1) or (10011). Addition is the same as XORing of 4-bit words. Figure P.11 shows the *MixColumns* transformation.

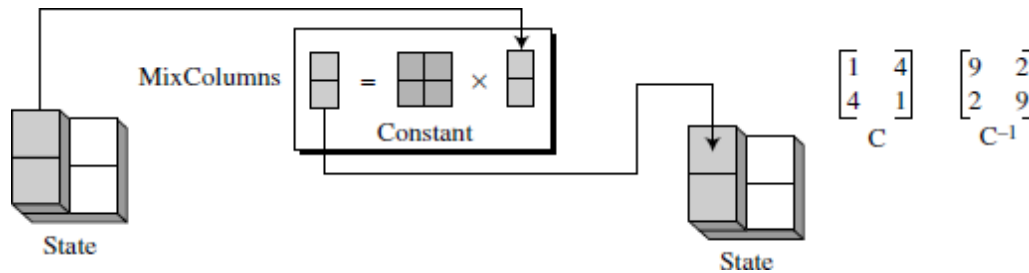


Figure P.11
MixColumns transformation

Key Adding

Probably the most important transformation is the one that includes the cipher key. All previous transformations use known algorithms that are invertible. If the cipher key is not added to the state at each round, it is very easy for the adversary to find the plaintext, given the ciphertext. The cipher key is the only secret between Alice and Bob in this case.

ES uses a process called key expansion (discussed later in this appendix) that creates three round keys from the cipher key. Each round key is 16 bits long

it is treated as two 8-bit words. For the purpose of adding the key to the state, each word is considered as a column matrix.

AddRoundKey

AddRoundKey also proceeds one column at a time. It is similar to *MixColumns* in this respect. *MixColumns* multiplies a constant square matrix by each state column. *AddRoundKey* adds a round key word with each state column matrix. The operations in *MixColumns* are matrix multiplication; the operations in *AddRoundKey* are matrix addition. The addition is performed in the GF(2⁴) field. Because addition and subtraction in this field are the same, the *AddRoundKey* transformation is the inverse of itself.

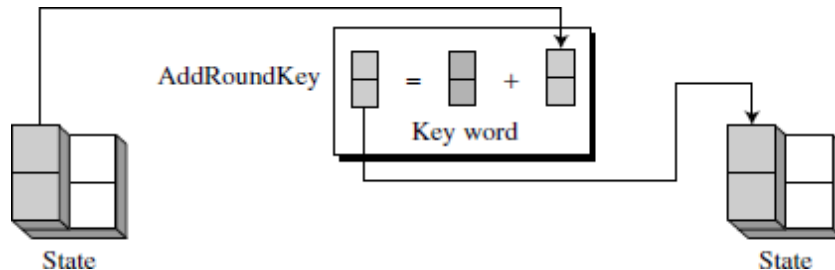


Figure P.13 shows the *AddRoundKey* transformation.

You take the following aes steps of encryption for a 128-bit block:

1. Derive the set of round keys from the cipher key.
2. Initialize the state array with the block data (plaintext).
3. Add the initial round key to the starting state array.
4. Perform nine rounds of state manipulation.
5. Perform the tenth and final round of state manipulation.
6. Copy the final state array out as the encrypted data (ciphertext)

Conclusion:

Thus we have implemented AES and study the encryption and decryption process.

The purpose was to create an algorithm that was resistant against known attacks, simple, and quick to code.

Assignment No : 5	Date:
Title: Implementation of RSA	
Sign:	Remark:

Title: Implementation of RSA**Problem Definition:** Write a Java / Python program for performs the RSA Algorithm.**Prerequisite:**

- Basic concepts of Java /Python

Tools/Framework/Language Used: Java / Python**Learning Objectives:** Understand the implementation of RSA.**Outcomes:** After completion of this assignment students will understand how RSA Algorithm works. The RSA algorithm's security is based on the difficulty of factoring large numbers into their prime factors.**Theory:****Concepts:**

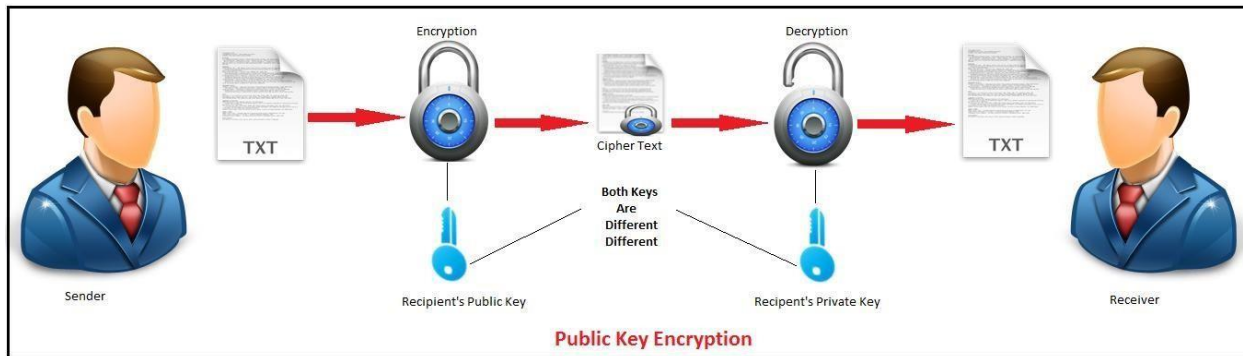
Data security and confidentiality issues are an important aspect of information systems. Because of the confidentiality and security of data, it is essential always to be concerned both for common security, as well as for the privacy of individuals. In this case, it is closely related to how important the information and data is sent and received by the interested person. Information is no longer useful if the information is tapped by an irresponsible person. For the data is not known by parties who are not concerned, then each data owner is always trying to do security with some specific techniques.

Cryptography is a science or art of securing a message and done by a cryptographer. While cryptanalysis is a science and art of solving ciphertext to plaintext. The person doing it is called cryptanalyst. The word cryptography comes from the Greek word "kryptos" which means to hide and "graphein" which means to write. Cryptography can be defined as a science that transforms information from understandable forms into incomprehensible forms. Cryptography Algorithm always consists of two parts, namely encryption, and decryption. Encryption is a process done to convert a readable message into an unreadable message (ciphertext). Decryption is the opposite of the encryption process, returning unread messages to readable messages. The encryption and decryption process is governed by one or more cryptographic keys. In a system where there is a cryptographic algorithm, plus all possible plaintext, ciphertext and keys are called cryptosystems.

Public-Key one of the major difficulties of conventional encryption is the need for security to distribute keys used in a secure state. The next generation technique is a modern cryptography. It is a way to remove this weakness with an encryption model that does not require a key to be distributed. This method is known as the "public-key" encryption and was first introduced in 1976. For conventional encryption, the keys used in the encryption and decryption process are the same. However, this is not necessary conditions. However, it is possible to construct an algorithm that uses one key for encryption and decryption. This process requires different keys for encryption and decryption. Furthermore, it is possible to create an algorithm in which the knowledge of the encryption algorithm and the encryption key is insufficient to determine the decryption key

RSA Algorithm is used to encrypt and decrypt data in modern computer systems and other electronic devices. RSA algorithm is an asymmetric cryptographic algorithm as it creates 2 different keys for the purpose of encryption and decryption. It is public key cryptography as one of the keys involved is made public. RSA stands for Ron Rivest, Adi Shamir and Leonard Adleman who first publicly described it in 1978.

RSA makes use of prime numbers (arbitrary large numbers) to function. The public key is made available publicly (means to everyone) and only the person having the private key with them can decrypt the original message.



The RSA algorithm involves three steps:

1. Key generation
2. Encryption
3. Decryption.

Key generation

For the RSA cryptosystem, we first start off by generating two large prime numbers, 'p' and 'q', of about the same size in bits. Next, compute 'n' where $n = pq$, and 'x' such that, $x = (p - 1)(q - 1)$. We select a small odd integer less than x, which is relatively prime to it i.e. $\gcd(e, x) = 1$. Finally we find out the unique multiplicative inverse of e modulo x, and name it 'd'. In other words, $ed = 1 \pmod{x}$, and of course, $1 < d < x$. Now, the public key is the pair (e,n) and the private key is d.

RSA Encryption

Suppose Bob wishes to send a message (say 'm') to Alice. To encrypt the message using the RSA encryption scheme, Bob must obtain Alice's public key pair (e,n). The message to send must now be encrypted using this pair (e,n). However, the message 'm' must be represented as an integer in the interval $[0, n-1]$. To encrypt it, Bob simply computes the number 'c' where $c = m^e \pmod{n}$. Bob sends the ciphertext c to Alice.

RSA Decryption

To decrypt the ciphertext c, Alice needs to use her own private key d (the decryption exponent) and the modulus n. Simply computing the value of $c^d \pmod{n}$ yields back the decrypted message (m). Any article treating the RSA algorithm in considerable depth proves the correctness of the decryption algorithm.

Working of RSA Algorithm

RSA involves use of public and private key for its operation. The keys are generated using the following steps:-

1. Two prime numbers are selected as **p** and **q**
2. **n = pq** which is the modulus of both the keys.
3. Calculate **totient = (p-1)(q-1)**
4. Choose **e** such that **e > 1** and coprime to **totient** which means **gcd (e, totient)** must be equal to **1**, **e** is the public key
5. Choose **d** such that it satisfies the equation **de = 1 + k (totient)**, **d** is the private key not known to everyone.
6. Cipher text is calculated using the equation **c = m^e mod n** where **m** is the message.
7. With the help of **c** and **d** we decrypt message using equation **m = c^d mod n** where **d** is the private key.

CONCLUSION

By using RSA Algorithm, a system capable of ensuring data confidentiality can be established. From a technical point of view, RSA has easy and simple encryption.

Assignment No : 6	Date:
Title: Implementation of Diffie-Hellman key exchange	
Sign:	Remark:

Title: Implementation of Diffie-Hellman key exchange

Problem Definition: Write a Java / Python/C++ program for performs the key exchange by using Diffie-Hellman Algorithm.

Prerequisite:

- Basic concepts of Java /Python/C++

Tools/Framework/Language Used: Java / Python/C++

Learning Objectives: Understand the implementation of Diffie-Hellman key exchange

Outcomes: After completion of this assignment students will understand how exchange key and mod operation while exchange of and Diffie-Hellman algorithm.

Theory:

Concepts:

The Diffie–Hellman key exchange algorithm solves the following dilemma. Alice and Bob want to share a secret key for use in a symmetric cipher, but their only means of communication is insecure. Every piece of information that they exchange is observed by their adversary Eve. How is it possible for Alice and Bob to share a key without making it available to Eve? At first glance it appears that Alice and Bob face an impossible task. It was a brilliant insight of Diffie and Hellman that the difficulty of the discrete logarithm problem for $F * p$ provides a possible solution.

The first step is for Alice and Bob to agree on a large prime p and a nonzero integer g modulo p . Alice and Bob make the values of p and g public knowledge; for example, they might post the values on their web sites, so Eve knows them, too. For various reasons to be discussed later, it is best if they choose g such that its order in $F * p$ is a large prime.

The next step is for Alice to pick a secret integer a that she does not reveal to anyone, while at the same time Bob picks an integer b that he keeps secret. Bob and Alice use their secret integers to compute

$$\underbrace{A \equiv g^a \pmod{p}}_{\text{Alice computes this}} \quad \text{and} \quad \underbrace{B \equiv g^b \pmod{p}}_{\text{Bob computes this}}.$$

They next exchange these computed values, Alice sends A to Bob and Bob sends B to Alice. Note that Eve gets to see the values of A and B , since they are sent over the insecure communication channel. Finally, Bob and Alice again use their secret integers to compute

$$\underbrace{A' \equiv B^a \pmod{p}}_{\text{Alice computes this}} \quad \text{and} \quad \underbrace{B' \equiv A^b \pmod{p}}_{\text{Bob computes this}}.$$

The values that they compute, A' and B' respectively, are actually the same, since $A' \equiv B^a \equiv (g^b)^a \equiv g^{ab} \equiv (g^a)^b \equiv A^b \equiv B' \pmod{p}$. This common value is their exchanged key. The Diffie–Hellman key exchange algorithm is summarized in Table 1.

Public Parameter Creation	
A trusted party chooses and publishes a (large) prime p and an integer g having large prime order in \mathbb{F}_p^* .	
Private Computations	
Alice	Bob
Choose a secret integer a . Compute $A \equiv g^a \pmod{p}$.	Choose a secret integer b . Compute $B \equiv g^b \pmod{p}$.
Public Exchange of Values	
<div style="display: flex; justify-content: space-between; align-items: center;"> <div style="text-align: center;"> <p>Alice sends A to Bob</p> <p>B</p> </div> <div style="flex-grow: 1; text-align: center;"> $\xrightarrow{\hspace{10em}}$ $\xleftarrow{\hspace{10em}}$ </div> <div style="text-align: center;"> <p>A</p> <p>Bob sends B to Alice</p> </div> </div>	
Further Private Computations	
Alice	Bob
Compute the number $B^a \pmod{p}$. The shared secret value is $B^a \equiv (g^b)^a \equiv g^{ab} \equiv (g^a)^b \equiv A^b \pmod{p}$.	Compute the number $A^b \pmod{p}$.

Table 1. Diffie–Hellman key exchange

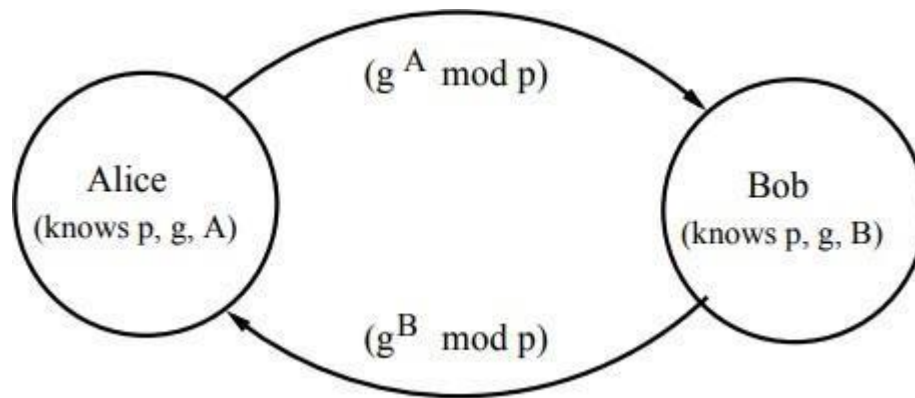


Figure 1: process of key exchange

Steps in the algorithm:

- 1 Alice and Bob agree on a prime number p and a base g .
- 2 Alice chooses a secret number a , and sends Bob $(g^a \bmod p)$.
- 3 Bob chooses a secret number b , and sends Alice $(g^b \bmod p)$.
- 4 Alice computes $((g^b \bmod p)^a \bmod p)$.
- 5 Bob computes $((g^a \bmod p)^b \bmod p)$. Both Alice and Bob can use this number as their key. Notice that p and g need not be protected

- 1 Alice and Bob agree on $p = 23$ and $g = 5$.
- 2 Alice chooses $a = 6$ and sends $5^6 \bmod 23 = 8$.
- 3 Bob chooses $b = 15$ and sends $5^{15} \bmod 23 = 19$.
- 4 Alice computes $19^6 \bmod 23 = 2$.
- 5 Bob computes $8^{15} \bmod 23 = 2$.

Then 2 is the shared secret.

Clearly, much larger values of a , b , and p are required. An eavesdropper cannot discover this value even if she knows p and g and can obtain each of the messages. Suppose p is a prime of around 300 digits, and a and b at least 100 digits each. Discovering the shared secret given g , p , $g^a \bmod p$ and $g^b \bmod p$ would take longer than the lifetime of the universe, using the best known algorithm. This is called the discrete logarithm problem.

How can two parties agree on a secret value when all of their messages might be overheard by an eavesdropper? The Diffie-Hellman algorithm accomplishes this, and is still widely used. With sufficiently large inputs, Diffie-Hellman is very secure.

Conclusion:

The Diffie-Hellman key exchange exploits mathematical properties to produce a common computational result between two (or more) parties wishing to exchange information, without any of them providing all the necessary variables. The Shared secret key is generated using diffie-hellman algorithm using authentication to provide more security.