SAVITRIBAI PHULE PUNE UNIVERSITY

FOURTH YEAR OF COMPUTER ENGINEERING (2015 COURSE)

410254: LABORATORY PRACTICE III

Teaching Scheme: Practical : 04 Hours/Week **Credit : 02**

Examination Scheme: 50 Marks Practical **Term Work:: 50 Marks**

LAB MANUAL OF

410251: INFORMATION AND CYBER SECURITY

ASSIGNMENT NO: 1

TITLE: To implement S-DES Algorithm.

OBJECTIVES:

1. To study S-DES Algorithm
2. To implement S-DES Algorithm

THEORY:

S-DES or Simplified Data Encryption Standard

S-DES is symmetric block cipher algorithm. The process of encrypting a plan text into an encrypted message with the use of S-DES has been divided into multi-steps which may help you to understand it as easily as possible.

Points should be remembered.

1. It is a block cipher.
2. It has 8-bits block size of plain text or cipher text.
3. It uses 10-bits key size for encryption.
4. It is a symmetric cipher.
5. It has Two Rounds.

- **Key Generation of S-DES or How to Generate the Key of Simplified DES?**

First and foremost, we need to generate a key. With the help of this key we will encrypt the message.
Now the interesting question is, how to generate the key, and where the key is to be used. Just follow the steps.

Step 1:

Just select a random key of 10-bits, which only should be shared between both parties which means sender and receiver.
As I selected below!

Select key:1010000010

Note: You can select any random number of 10-bits.

Step 2:

Put this key into P.10 Table and permute the bits.

P.10 Table:

Input	1	2	3	4	5	6	7	8	9	10
Output Should be	3	5	2	7	4	10	1	9	8	6

As I put key into P.10 Table.

Input	1	0	1	0	0	0	0	0	1	0
Output	1	0	0	0	0	0	1	1	0	0

Now the output will be:
Key: 1000001100

Step 3:
Divide the key into two halves, left half and right half;
{1 0 0 0 0} | {0 1 1 0 0}

Step 4:
Now apply the one bit Round shift on each half:
Before round shift: {10000} | {01100}
After round shift: {00001} | {11000}
The output will be:
{0 0 0 0 1} {1 1 0 0 0}

Step 5:
Now once again combine both halve of the bits, right and left. Put them into the P8 table. What you get, that will be the K1 or First key.
Combine: 0 0 0 0 1 1 1 0 0 0

Permute into 8bit table:
P8-Table

Input	1	2	3	4	5	6	7	8	9	10
Combine-bits	0	0	0	0	1	1	1	0	0	0
Output Should be	6	3	7	4	8	5	10	9		
Output bits	1	0	1	0	0	1	0	0		

See the table the 1 and 2 number of bits are removed and other are permuted, as 6 in place of one, 9 in place of 8 and so on.

The output and K1 or key One will be:

K1=1 0 1 0 0 1 0 0

Step6:

As we know S-DES has two round and for that **we also need two keys**, one key we generate in the above steps (step 1 to step 5). Now we need to **generate a second** bit and after that we will move to encrypt the plain text or message.

It is simple to generate the second key. Simply, go in **step 4** copy both halves, each one consists of 5 bits. But be careful on the taking of bits. Select those halves which are output of first round shift, don’t take the bits which are not used in the first round. In simple words, take the output of first round shift in above **step 4.**

Which are: {00001} | {11000}

Step 7:

Now just apply two round shift circulate on each half of the bits, which means to change the position of two bits of each halves.
left half: 00001
Right half: 11000

After the two rounds shift on each half out-put of each half will be.
Left half: 00100
Right half: 00011
Combine both together: As: 0 0 1 0 0 – 0 0 0 1 1

Step 8:

Now put the bits into 8-P Table, what you get, that will be your second key. Table is also given in **step 5**.

But here the combinations of bits are changed because of two left round shift from step 5. Check it in depth.
Combine bits: 0 0 1 0 0 0 0 0 1 1

P.8 Table

Input	1	2	3	4	5	6	7	8	9	10
Combine-bits	0	0	1	0	0	0	0	0	1	1
Output Should be	6	3	7	4	8	5	10	9		
Output bits	0	1	0	0	0	0	1	1		

The output of the bits are your Second key or K2:
K2: 0 1 0 0 0 0 1 1

Finally we create both keys successfully:

K1: 1 0 1 0 0 1 0 0 (see in step 5)
K2: 0 1 0 0 0 0 1 1 (see in step 8)

How To Encrypt the Plain Text into Cipher Text in S-DES After Generating Keys.?

Now, let’s start Encryption of plain text into cipher text.

Encryption of Plain text into Cipher text in S-DES:

Come on do it, **step by step**.
Note: the size of input text is 8 bit and output also will be 8-bit. Or the block size is 8-bit/one byte always.

Step 1:

Suppose this is our plain text in binary which is 8-bit.

Plain text: 01110010

Step 2:

Put the plain text into IP-8(initial permutation) table and permute the bits.

IP-8 table

Bits number	1	2	3	4	5	6	7	8
Bits to be permuted	0	1	1	1	0	0	1	0
Permute the bits	2	6	3	1	4	8	5	7
Permuted bits	1	0	1	0	1	0	0	1

Output is: 1 0 1 0 1 0 0 1

Step 3:

Now break the bits into two halves, each half will consist of 4 bits. The halves will be right and left.
Two Halves of the bits:

Left half {1 0 1 0} -right half {1 0 0 1}

Step 4:

Take the right 4 bits and put them into E.P (expand and per-mutate) Table.
Bits of right half: 1001

E.P Table

Right half bits	1	0	0	0				
Number	1	2	3	4	5	6	7	8
Expand bits	4	1	2	3	2	3	4	1
Output of bits	0	1	0	0	0	0	0	1

Output of right four bits will be 8 bits, after their expanding with the help of E.P table.

O-P: 0 1 0 0 0 0 0 1

Step 5: Now, just take the output and XOR it with First key Or K 1 (which we created in previous topic that is how to generate key.).

XOR (⊕) them:

O-p: 0 1 0 0 0 0 0 1
⊕
K1: <u>1 0 1 0 0 1 0 0</u>
Output of XOR: **1 1 1 0 0101**

Step 6:

Once again split the output of XOR's bit into two halves and each half will consist of 4 bits.

Splitting them into two halves:

Left half :{ 1 1 1 0} right half :{ 0101}
Now put the each half into the **s-boxes**, there is only two s-boxes. **S-0 and S-1.**

S-0

Col	0	1	2	3
Rows				
0	01	00	11	10
1	11	10	01	00
2	00	10	01	11
3	11	01	11	10

S-1

Col	0	1	2	3
Rows				
0	00	01	10	11

1	10	00	01	11
2	11	00	01	00
3	10	01	00	11

Note: put the left half into S-0 box and put the right half into S-1 Box.

But how to put them in into S-Boxes?

Take any half, (but don't forget the above mentioned note..).
The most first and most last bit will be consider the row and other remaining, which are, 2 and 3, will be considered the columns.
See, here I'm taking the left half: which is 1 1 1 0.

Now I will take First and last bit which are: 1 and 0. These will be row.

And I also will take 2nd and 3rd bits which are: 11. These will be column number.
10 means=2nd **row**
11 means = 3rd **col**

See below how it will be the second row, and 3rd column. Please, remember the IP Addressing, such as 2^8 for 255.
$1^2\, 0^1 = 2+0=2$
$1^2\, 1^1 = 2+1=3$

Let's check the 2nd row and 3rd column.

For **left half** we check the in S-**0** . In which 2nd row and 3rd column.

The output is 00 for left half;

Now let's take the **right half** and find the output of the right of in **S-1 box**.
Right half: 0101
Row: 0 1= $0^2\, 1^1=0+1=1$
Col: 1 0=$1^2\, 0^1=2+1=3$

Which means the value will be in 1st row and 3rd column, let's in **check S-1.**
The output will be: 11 for left half.

Step 7:
Now combine these two halves together.

Left half: {00} and right half: {11}
It will be: 0 0 1 1

Step 8: Now take these 4 bits and put them in **P-4** (permutation 4) table and get the result.

P 4 Table

Numbers	1	2	3	4
Input	0	0	1	1
Output should be	2	4	3	1
Out-Put	0	1	1	0

Now the output is: 0 1 1 0

Step 9:

Now get XOR the output with left 4 bits of Initial Per-mutation. The left bits of initial per-mutation are in **step 3**, which are **1 0 1 0**.(please, in step 3).
Let them to be XOR.

0 1 1 0
$\oplus$
$\underline{1\ 0\ 1\ 0}$

Out-put will be: 1 1 0 0

Step 10:

Now get the right half of the initial permutation, which is **step 3,** and combine that with this out-put.

The out-put of XOR in **step 9:** 1 1 0 0
Right half of IP (initial permutation): 1 0 0 1
Let's combine both. 1 1 0 0 – 1 0 0 1= 1 1 0 0 1 0 0 1
Now the output is 8 bits.: **1 1 0 0 1 0 0 1**

Step 11: Now once again break the out-put into two halves, left and right;

Left: {1 1 0 0} right: {1 0 0 1}
Step 12:
Now swap both halves, which means put the left half in place of right and vice versa.
Result:
Left half: {1 0 0 1} right half: {1 1 0 0}

Step 13:

Now let's take these halves and once again start the same procedure from **step 2** or initial Permutation, **BUT** be careful on using key in this stage we use second key or K2 (not K1). And put that into IP^{-1} (IP inverse) Table. What you get will be your final cipher text.

Let me to do it in brief. You should check carefully what I did.
Ø 1 0 0 1 11 0 0

After initial permutation according to **IP-8 table (see step 2)**
Out-put will be: 0 1 0 1 1 0 1 0

Ø Now break it into two halves
Left {0 1 0 1} right {1 0 1 0}

Ø Now Take the right 4bits and put them into **EP table**, and get the result of 8 bits.
It will be: 0 1 0 1 0 1 0 1

Ø Let **XOR it with K2.**
K2: 0 1 0 0 0 0 1 1
$\oplus$

Out-put of **EP**: <u>0 1 0 1 0 1 0 1</u>
Out- Put: **0 0 0 1 0 1 1 0**

Ø Once again split the output of XOR’s bit into two halves:
Left: {0 0 0 1} right: {0 1 1 0}

Ø Now put each half in S-Boxes, which are S-0 and S-1:

Note: put the left half into S-0 box and put the right half into S-1 Box.

Before that get Rows and column:

Left: 0 0 0 1
Row: 0 1= 1
Col: 0 0 = 0

Let find the row and column of left, in S-0, the value is row number one and col number Zero.
It will be. 1 1
Ø now check the right half: 0 1 1 0
Row: 0 0=0
Col: 1 1= 3

Let's find the row and column of right, in S-1, the value is row number zero and col number three.

It will be: 11
Ø Now combine both halves together.
It will be: 1 1 1 1

Ø Now take these 4 bits and put them in **P-4** (**Per-mutation** 4) table and get the result.

The out-put also will be: 1111 after permutation.

Ø Now get it XOR with left 4 bits of Initial Per-Mutation.
1 1 1 1
XOR
0 1 0 1

Out-put:1 0 1 0

Ø Now get the right half of the initial permutation and combine that with this out- put.
1 0 1 0 - 0 1 1 0

Ø Now once again break the it into two halves, left and right
Left: {1 0 1 0} right: {0 1 1 0}

Ø Now **swap both halves,** which means put the left half in place of right and vice versa.
0 1 1 0 1 0 1 0

Ø Now put it into IP^{-1} Table which is :

IP^{-1} Table

Numbers	1	2	3	4	5	6	7	8
input	0	1	1	0	1	0	1	0
Out-put to be	2	6	3	1	4	8	5	7
Out-Put	1	0	1	0	0	0	1	1

Out-Put is the cipher text. Which is: 1 0 1 0 0 0 1 1

Finally we Encrypted successfully our **plain text: 01110010** into **cipher text** which is:

1 0 1 0 0 0 1 1

CONCLUSION:

Thus we have successfully implemented Simplified DES algorithm**.**

ASSIGNMENT NO: 2

TITLE: To implement S-AES Algorithm.

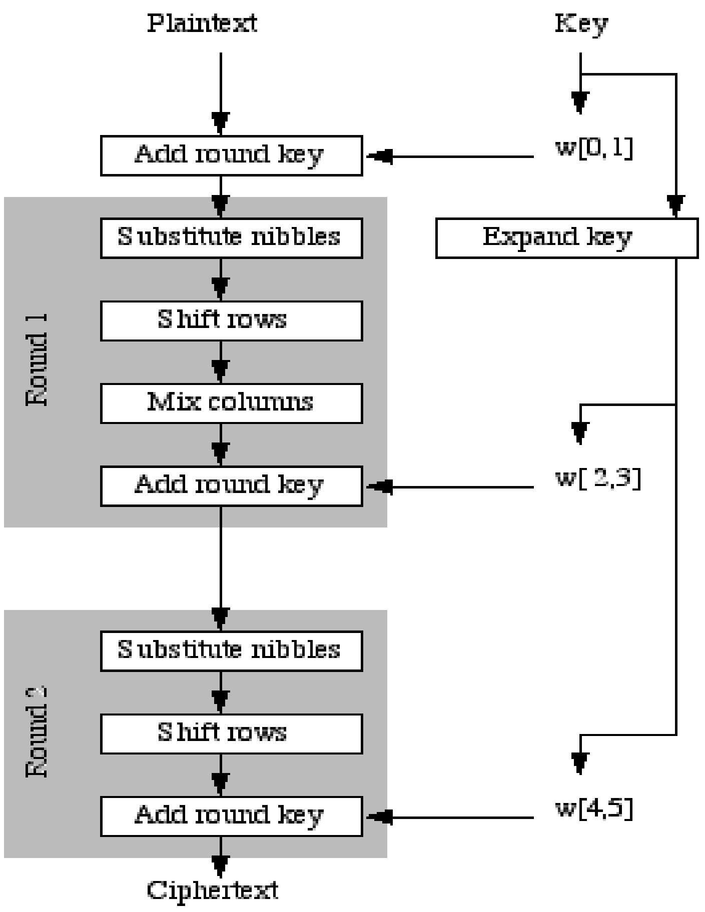
OBJECTIVES:

1. To study S-AES Algorithm
2. To implement S-AES Algorithm

THEORY:

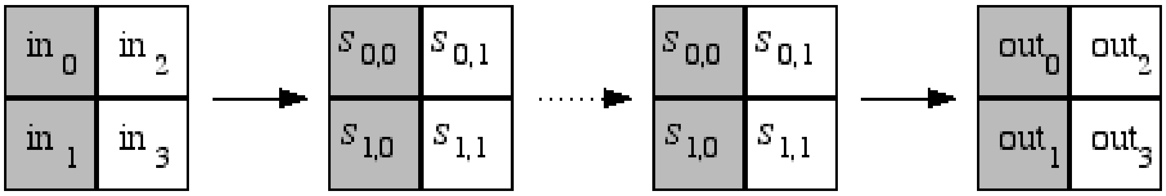
A Simplified AES Algorithm:

S-AES is to AES as S-DES is to DES. In fact, the structure of S-AES is exactly the same as AES. The differences are in the key size (16 bits), the block size (16 bits) and the number of rounds (2 rounds). Here is an overview:



S-AES Encryption Overview

Substitute nibbles Instead of dividing the block into a four by four array of bytes, S-AES divides it into a two by two array of "nibbles", which are four bits long. This is called the state array and is shown below.



S-AES state array

In the first stage of each encryption round, an S-box is used to translate each nibble into a new nibble. First we associate the nibble $b_0b_1b_2b_3$ with the polynomial $b_0x^3 + b_1x^2 + b_2x + b_3$. This polynomial is then inverted as an element of GF(16), with the "prime polynomial" used being $x^4 + x + 1$. Then we multiply by a matrix and add a vector as in AES.

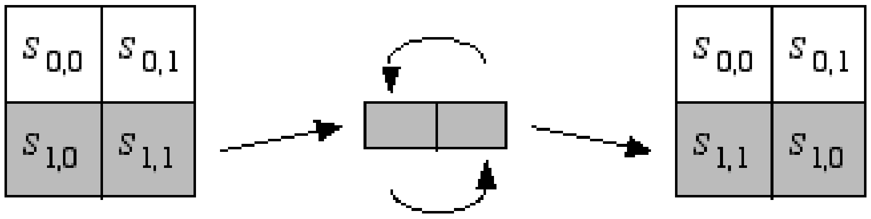
$$\begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

Remember that the addition and multiplication in the equation above are being done modulo 2 (with XOR), but not in GF(16).

Since a computer would do the S-box substitution using a table lookup, we give the full table for the S-box here.

nibble	S-box(nibble)	nibble	S-box(nibble)
0000	1001	1000	0110
0001	0100	1001	0010
0010	1010	1010	0000
0011	1011	1011	0011
0100	1101	1100	1100
0101	0001	1101	1110
0110	1000	1110	1111
0111	0101	1111	0111

Shift Rows The next stage is to shift the rows. In fact, the first row is left alone and the second row is shifted.



S-AES shift row transformation

Mix Columns: After shifting the rows, we mix the columns. Each column is multiplied by the matrix.

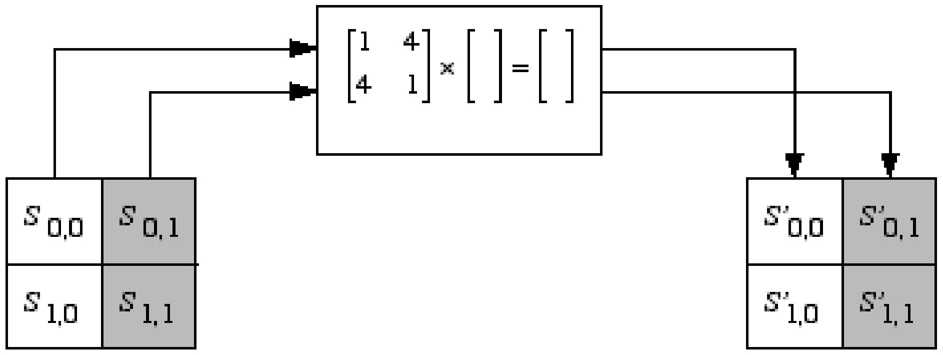
$$\begin{bmatrix} 1 & 4 \\ 4 & 1 \end{bmatrix}$$

These operations **are** done in GF(16), so remember that the 1 corresponds to the polynomial 1 and the 4 corresponds to the polynomial x^2. Thus this matrix could also be written as

$$\begin{bmatrix} 1 & x^2 \\ x^2 & 1 \end{bmatrix}$$

Don't forget to reduce modulo $x^4 + x + 1$.

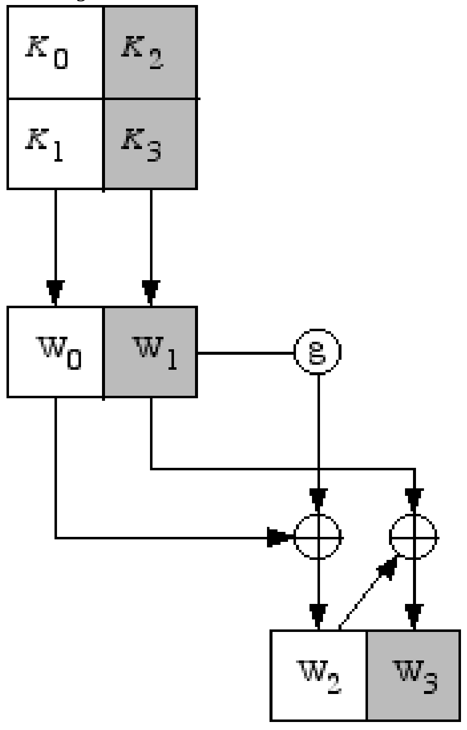
The mix column transformation is omitted in the last round, in order to simplify the decryption



S-AES mix column transformation

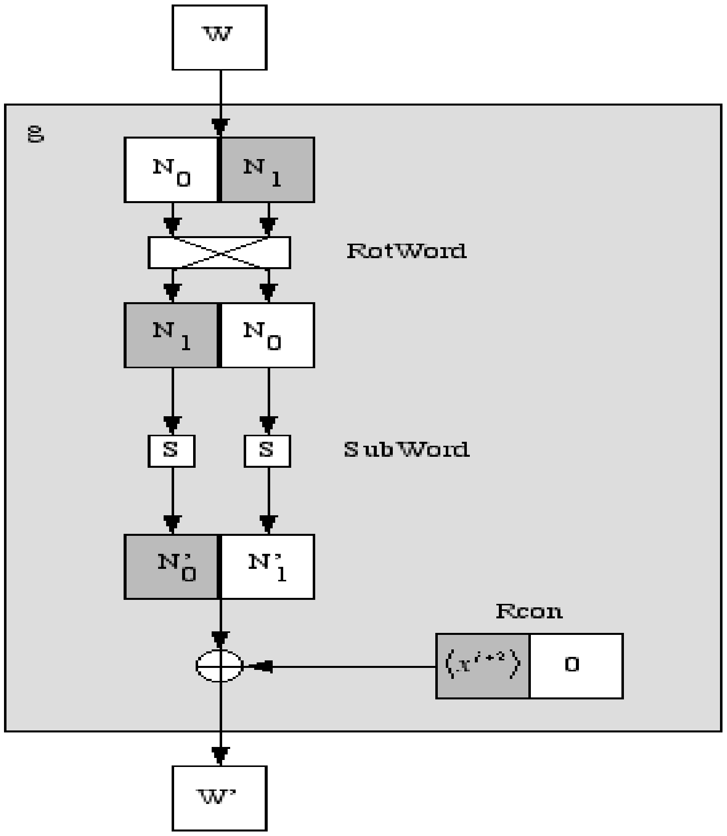
Add Round Key The last stage of each round of encryption is to add the round key. (In fact, this is also done before the first round.) Before the first round, the first two words (W_0 and W_1) of the expanded key are added. In the first round, W_2 and W_3 are added. In the last round, W_4 and W_5 are added. All additions are done modulo 2, that is, with XOR.

Key Expansion Key expansion is done very similarly to AES. The four nibbles in the key are grouped into two 8-bit "words", which will be expanded into 6 words. The first part of the expansion, which produces the third and fourth words, is shown below. The rest of the expansion is done in exactly the same way, replacing W_0 and W_1 with W_2 and W_3, and replacing W_2 and W_3 with W_4 and W_5.



S-AES key expansion

The g function is shown in the next diagram. It is very similar to AES, first rotating the nibbles and then putting them through the S-boxes. The main difference is that the round constant is produced using x^{j+2}, where j is the number of the round of expansion. That is, the first time you expand the key you use a round constant of x^3 = 1000 for the first nibble and 0000 for the second nibble. The second time you use x^4 = 0011 for the first nibble and 0000 for the second nibble.



Advantages of S-AES :

- This enables more rapid and effective encryption and decryption than would be possible with pure software solutions.
- The **AES** enjoys huge popularity because the **advantages** speak for themselves. For example this encryption standard is freely usable, incurs no license fees and is not subject to patent restrictions.

<u>CONCLUSION:</u>

Thus we have studied and implemented S-AES.

ASSIGNMENT NO: 3

TITLE:

To implement the Diffie-Hellman Key Exchange algorithm.

OBJECTIVE:

To study and implement Diffie-Hellman Key Exchange algorithm.

THEORY:

Diffie–Hellman Key Exchange establishes a shared secret between two parties that can be used for secret communication for exchanging data over a public network. It is primarily used as a method of exchanging cryptography keys for use in symmetric encryption algorithms like AES. The algorithm in itself is very simple.

The process begins by having the two parties, Alice and Bob. Let's assume that Alice wants to establish a shared secret with Bob.

EXAMPLE:

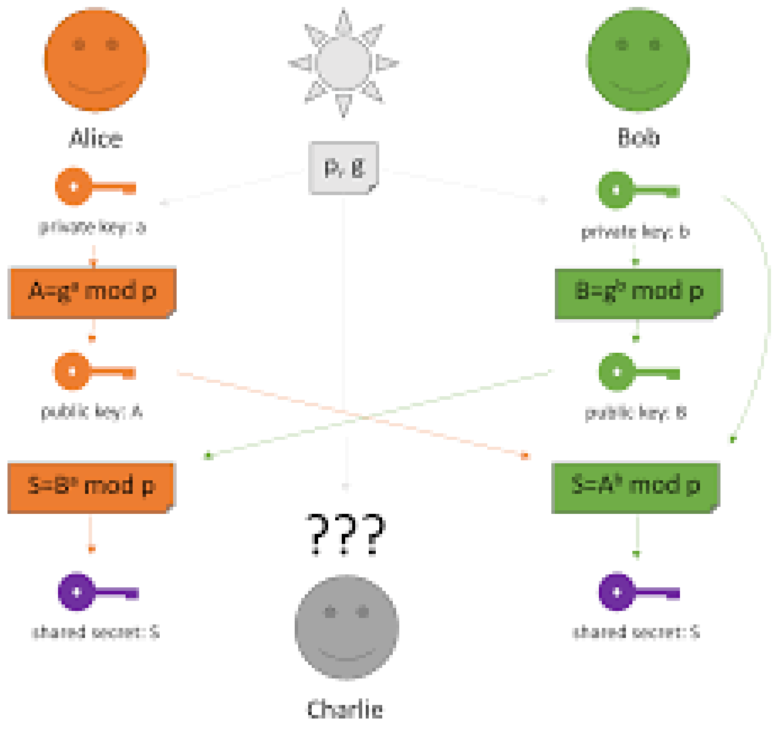


Figure: Working of Diffie-Hellman Key Exchange Algorithm

ALGORITHM:

STEP-1:

Both Alice and Bob shares the same public keys g and p.

STEP-2:

Alice selects a random public key a.

STEP-3:

Alice computes his secret key A as g^a mod p.

STEP-4:

Then Alice sends A to Bob.

STEP-5:

Similarly Bob also selects a public key b and computes his secret key as B and sends the same back to Alice.

STEP-6:

Now both of them compute their common secret key as the other one's secret key power of a mod p.

CONCLUSION:

Thus we have successfully implemented Diffie-Hellman Key Exchange algorithm

ASSIGNMENT NO: 4

TITLE: Implementation of RSA algorithm.

OBJECTIVE: To study,

2. Public key algorithm.
3. RSA algorithm
4. Concept of Public key and Private Key.

THEORY:

Public Key Algorithm:

Asymmetric algorithms rely on one key for encryption and a different but related key for decryption. These algorithms have the following important characteristics:

- It is computationally infeasible to determine the decryption key given only knowledge of the cryptographic algorithm and the encryption key.

In addition, some algorithms, such as RSA, also exhibit the following characteristics:

- Either of the two related keys can be used for encryption, with the other used for decryption.
  A public key encryption scheme has six ingredients:

- **Plaintext:** This is readable message or data that is fed into the algorithm as input.

- **Encryption algorithm:** The encryption algorithm performs various transformations on the plaintext.

- **Public and private key:** This is a pair of keys that have been selected so that if one is used for encryption, the other is used for decryption. The exact transformations performed by the algorithm depend on the public or private key that is provided as input.

- **Ciphertext:** This is the scrambled message produced as output. It depends on the plaintext and the key. For a given message, two different keys will produce two different ciphertexts.

- **Decryption algorithm:** This algorithm accepts the ciphertext and the matching key and produces the original plaintext.

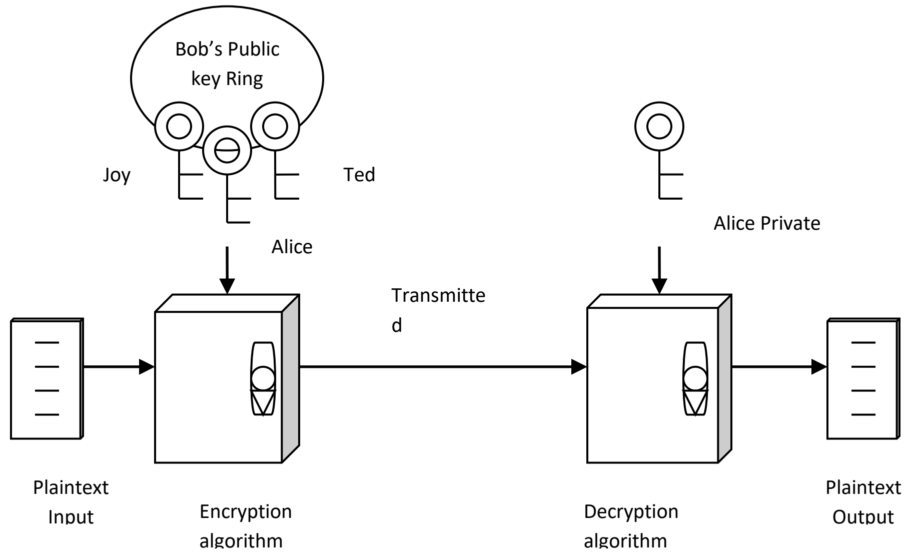


Figure: Public key cryptography

The essential steps are as the following:

1. Each user generates a pair of keys to be used for the encryption and decryption of messages.
2. Each user places one of the two keys in a public register or the other accessible file. This is the public key. The companion key is kept private. As figure suggests, each user maintains a collection of public keys obtained from others.
3. If Bob wishes to send a confidential message to Alice, Bob encrypts the message using Alice's public key.
4. When Alice receives the message, the decrypts it using her private key. No other recipient can decrypt the message because only Alice knows Alice's private key.

The RSA Algorithm:

The scheme developed by Rivest, Shamir and Adleman makes use of an expression with exponentials. Plaintext is encrypted in blocks, with each block having a binary value less than some number n. That is the block size must be less than or equal to log2 (n); in practice the block size is I bits, where 2i<n<=2i+1. Encryption and decryption are of the following form, for some plaintext block M and ciphertext block C:

$C = M^e \bmod n$

M = Cd mod n = (Me)d mod n = Med mod n

Both sender and receiver must know the value of n. The sender knows the value of e, and only the receiver knows the value of d. Thus, this is a public-key encryption algorithm with a public key of PU = {e, n} and a private key of PR = {d, n}. For this algorithm to be satisfactory for public key encryption, the following requirements must meet:

1. It possible to find values of e, d, n such that Med mod n = M for all M<n.
2. It is relatively easy to calculate Me mod n and Cd mod n for all values of M<n.
3. It is feasible to determine d given e and n.

Key Generation	
Select p, q	p and q both prime, p≠q
Calculate n = p * q	
Calculate Ø(n) = (p-1)(q-1)	
Select integer e	gcd(Ø(n),e) = 1; 1<e< Ø(n)

Encryption	
Plaintext	M<n

Decryption	
Ciphertext	C

Figure: The RSA Algorithm

Example:

1. Select two prime numbers, p = 17 and q = 11.
2. Calculate n = pq = 17*11 = 187.
3. Calculate Ø(n) = (p-1)(q-1) = 16*10 = 160.
4. Select e such that relatively prime to Ø(n) = 160 and less than Ø(n); we choose e = 7.
5. Determine d such that de ≡ 1 (mod 160) and d < 160. The correct value is d = 23, because 23*7 = 161 = 10*160+1; d can be calculated using the extended Euclid's algorithm.

The resulting keys are public key PU = {7, 187} and private key PR = {23, 187}. The example shows the use of these keys for plaintext input of M=88.

Advantages:

1. Easy to implement.

Disadvantages:

1. Any one can announce the public key.

Input:

- Two prime numbers p = 17 and q = 11.
- Select e = 7.
- Plaintext = 88.

Output:

- PU = 7, 187.
- PR = 23, 187.
- Ciphertext = 11.

ALGORITHM:

1. Start
2. Input two prime numbers p and q.
3. Calculate n = pq.
4. Calculate Ø(n) = (p-1)(q-1).
5. Input value of e.
6. Determine d.
7. Determine PU and PR.
8. Take input plaintext.
9. Encrypt the plaintext and show the output.
10. Stop.

CONCLUSION: We have studied and implemented the public key RSA algorithm.

ASSIGNMENT NO: 5

TITLE: Implementation of ECC algorithm.

OBJECTIVE: To study,

1. Public key algorithm.
2. ECC algorithm
3. Concept of Public key and Private Key.

THEORY:

Elliptic Curve Cryptography

Elliptic Curve Cryptography (ECC) was discovered in 1985 by Victor Miller (IBM) and Neil Koblitz (University of Washington) as an alternative mechanism for implementing public-key cryptography.

The security of elliptic curve cryptography depends on the ability to compute a point multiplication and the inability to compute the multiplicand given the original and product points. The size of the elliptic curve determines the difficulty of the problem.

The primary benefit promised by elliptic curve cryptography is a smaller key size, reducing storage and transmission requirements, i.e. that an elliptic curve group could provide the same level of security afforded by an RSA-based system with a large modulus and correspondingly larger key: for example, a 256-bit elliptic curve public key should provide comparable security to a 3072-bit RSA public key.

The equation of an elliptic curve is given as,

$$y^2 = x^3 + ax + b$$

Few terms that will be used,

E -> Elliptic Curve

P -> Point on the curve

n -> Maximum limit (This should be a prime number)

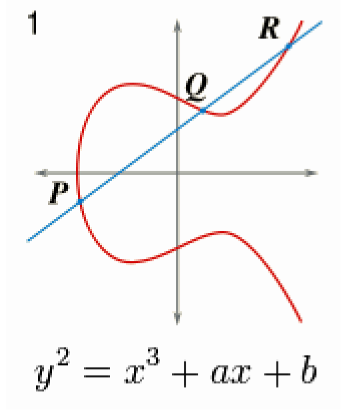


Fig 3

The fig 3 show are simple elliptic curve.

Key Generation

Key generation is an important part where we have to generate both public key and private key. The sender will be encrypting the message with receiver's public key and the receiver will decrypt its private key.

Now, we have to select a number **'d'** within the range of **'n'**.

Using the following equation we can generate the public key

Q = d * P

d = The random number that we have selected within the range of (**1 to n-1**). **P** is the point on the curve.

'Q' is the public key and **'d' is the private key.**

Encryption

Let 'm' be the message that we are sending. We have to represent this message on the curve. This have in-depth implementation details. All the advance research on ECC is done by a company called certicom.

Conside *'m'* has the point *'M'* on the curve *'E'*. Randomly select 'k' from [1 – (n-1)].

Two cipher texts will be generated let it be **C1** and **C2**.

C1 = k*P

C2 = M + k*Q

C1 and C2 will be send.

Decryption

We have to get back the message 'm' that was send to us,

M = C2 – d * C1

M is the original message that we have send.

Proof

How does we get back the message,

M = C2 – d * C1

'M' can be represented as 'C2 – d * C1'

C2 – d * C1 = (M + k * Q) – d * (k * P) (C2 = M + k * Q and C1 = k * P)

= M + k * d * P – d * k *P (canceling out k * d * P)

= M (Original Message)

Applications

Elliptic curves are applicable for encryption, digital signatures, pseudo-random generators and other tasks. They are also used in several integer factorization algorithms that have applications in cryptography, such as Lenstra elliptic curve factorization.

CONCLUSION: We have studied and implemented the public key ECC algorithm.