STES’s

NBN SINHGAD SCHOOL OF ENGINEERING

Ambegaon(Bk), Pune.

Department of Computer Engineering

LABORATORY MANUAL

2018-19

LABORATORY PRACTICE-III

BE-COMPUTER ENGINEERING

SEMESTER-II

INDEX

Assignment No.1

Aim: To implement Linear Regression to find the equation of the best fit line for given data.

Objective:

- The Basic Concepts of Linear Regression.
- Implementation logic of Linear Regression to find the equation of the best fit line for given data.

Theory:

Introduction:

Regression analysis is a very widely used statistical tool to establish a relationship model between two variables. One of these variable is called predictor variable whose value is gathered through experiments. The other variable is called response variable whose value is derived from the predictor variable.

Linear models are the simplest parametric methods and always deserve the right attention, because many problems, even intrinsically non-linear ones, can be easily solved with these models. A regression is a prediction where the target is continuous and its applications are several, so it's important to understand how a linear model can fit the data, what its strengths and weaknesses are, and when it's preferable to pick an alternative.

Types of Regression

- Linear
- Multiple Linear
- Logistic
- Polynomial

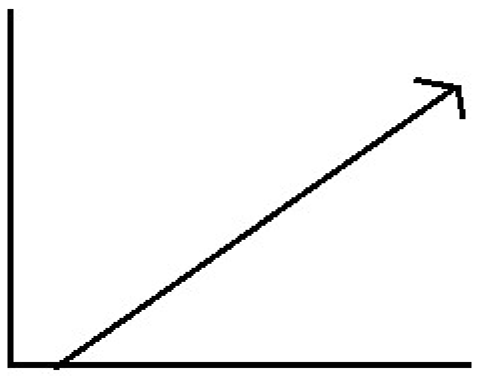
In Linear Regression these two variables are related through an equation, where exponent (power) of both these variables is 1. Mathematically a linear relationship represents a straight line when plotted as a graph. A non-linear relationship where the exponent of any variable is not equal to 1 creates a curve.

The general mathematical equation for a linear regression is –

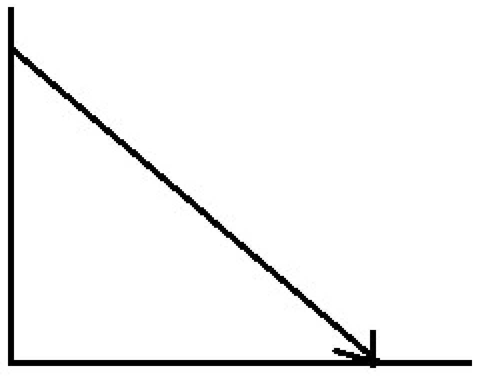
y = ax + b

y is the response variable and x is the predictor variable.

a and b are constants which are called the coefficients.



Positive Linear Relationship



Negative Linear Relationship

Applications

• **Trend lines:** A trend line represents the variation in some quantitative data with passage of time (like GDP, oil prices, etc.). These trends usually follow a linear relationship. Hence, linear regression can be applied to predict future values.

• **Economics:** To predict consumption spending, fixed investment spending, inventory investment, purchases of a country‘s exports, spending on imports, the demand to hold liquid assets, labor demand, and labor supply.

• **Finance:** Capital price asset model uses linear regression to analyze and quantify the systematic risks of an investment.

• **Biology:** Linear regression is used to model causal relationships between parameters in biological systems.

Python Packages needed

• pandas

– Data Analytics

• numpy

– Numerical Computing

• matplotlib.pyplot

– Plotting graphs

• sklearn

– Regression Classes

Steps to establish Linear Regression

A simple example of regression is predicting weight of a person when his height is known. To do this we need to have the relationship between height and weight of a person.

The steps to create the relationship is −

- Carry out the experiment of gathering a sample of observed values of height and corresponding weight.
- Create the object of Linear Regression Class.
- Train the algorithm with dataset of X and y.
- Get a summary of the relationship model to know the average error in prediction. Also called residuals.
- To predict the weight of new persons, use the predict() function.

Conclusion: We have studied the Linear Regression and also implemented successfully.

Assignment No. 2

Aim: To implement Decision Tree Classifier.

Objective:

- The Basic Concepts of Decision Tree Classifier.
- Implementation logic of Decision Tree Classifier.

Theory:

Introduction:

A decision tree is a flowchart-like structure in which internal node represents a "test" on an attribute (e.g. whether a coin flip comes up heads or tails), each branch represents the outcome of the test and each leaf node represents a class label (decision taken after computing all attributes). The paths from root to leaf represent classification rules.

Decision trees are diagrams that attempt to display the range of possible outcomes and subsequent decisions made after an initial decision. For example, your original decision might be whether to attend college, and the tree might attempt to show how much time would be spent doing different activities and your earning power based on your decision. There are several notable pros and cons to using decision trees.

In decision analysis, a decision tree and the closely related influence diagram are used as a visual and analytical decision support tool, where the expected values (or expected utility) of competing alternatives are calculated.

A decision tree consists of 3 types of nodes:

1. **Decision nodes** - commonly represented by squares
2. **Chance nodes** - represented by circles
3. **End nodes** - represented by triangles

Decision trees are commonly used in operations research, specifically in decision analysis, to help identify a strategy most likely to reach a goal. If in practice decisions have to be taken online with no recall under incomplete knowledge, a decision tree should be paralleled by a probability model as a best choice model or online selection model algorithm. Another use of decision trees is as a descriptive means for calculating conditional probabilities.

Decision trees, influence diagrams, utility functions, and other decision analysis tools and methods are taught to undergraduate students in schools of business, health economics, and public health, and are examples of operations research or management science methods.

ALGORITHM USED:

In decision tree learning, ID3 (Iterative Dichotomiser 3) is an algorithm invented by Ross Quinlan used to generate a decision tree from a dataset.

The ID3 algorithm begins with the original set as the root node. On each iteration of the algorithm, it iterates through every unused attribute of the set and calculates the entropy (or information gain) of that attribute. It then selects the attribute which has the smallest entropy (or largest information gain) value. The set is then split by the selected attribute (e.g. age < 50, 50 <= age < 100, age >= 100) to produce subsets of the data. The algorithm continues to Recurse on each subset, considering only attributes never selected before.

Recursion on a subset may stop in one of these cases:

- Every element in the subset belongs to the same class (+ or -), then the node is turned into a leaf and labelled with the class of the examples
- There are no more attributes to be selected, but the examples still do not belong to the same class (some are + and some are -), then the node is turned into a leaf and labelled with the most common class of the examples in the subset
- There are no examples in the subset, this happens when no example in the parent set was found to be matching a specific value of the selected attribute, for example if there was no example with age >= 100. Then a leaf is created, and labelled with the most common class of the examples in the parent set.

Throughout the algorithm, the decision tree is constructed with each non-terminal node representing the selected attribute on which the data was split, and terminal nodes representing the class label of the final subset of this branch.

Summary:

1. Calculate the entropy of every attribute using the data set

2. Split the set into subsets using the attribute for which entropy is minimum (or, equivalently, information gain is maximum)

3. Make a decision tree node containing that attribute.

4. Recurse on subsets using remaining attributes

Decision trees offer **advantages** over other methods of analyzing alternatives. They are:

- **Graphic**

  You can represent decision alternatives, possible outcomes, and chance events schematically. The visual approach is particularly helpful in comprehending sequential decisions and outcome dependencies.

- **Efficient**

  You can quickly express complex alternatives clearly. You can easily modify a decision tree as new information becomes available. Set up a decision tree to compare how changing input values affect various decision alternatives. Standard decision tree notation is easy to adopt.

- **Revealing**

  You can compare competing alternatives-even without complete information-in terms of risk and probable value. The Expected Value (EV) term combines relative investment costs, anticipated payoffs, and uncertainties into a single numerical value. The EV reveals the overall merits of competing alternatives.

- **Complementary**

  You can use decision trees in conjunction with other project management tools. For example, the decision tree method can help evaluate project schedules.

- Decision trees are self-explanatory and when compacted they are also easy to follow. In other words if the decision trees has a reasonable number of leaves, it can be grasped by non-professional users. Furthermore decision trees can be converted to a set of rules. Thus, this representation is considered as comprehensible.
- Decision trees can handle both nominal and numerical attributes.
- Decision trees representation is rich enough to represent any discrete-value classifier.
- Decision trees are capable of handling datasets that may have errors.
- Decision trees are capable of handling datasets that may have missing values.
- Decision trees are considered to be a nonparametric method. This means that decision trees have no assumptions about the space distribution and the classifier structure.

On the other hand, decision trees have **disadvantages** such as:

- Most of the algorithms (like ID3 and C4.5) require that the target attribute will have only discrete values.
- As decision trees use the ―divide and conquer‖ method, they tend to perform well if a few highly relevant attributes exist, but less so if many complex interactions are present. One of the reasons for this is that other classifiers can compactly describe a classifier that would be very challenging to represent using a decision tree.
- The greedy characteristic of decision trees leads to another disadvantage that should be pointed out. This is its over-sensitivity to the training set, to irrelevant attributes and to noise.

Conclusion: We have studied the Decision tree classifier and also implemented successfully.

Assignment No.3

Aim: To implement k-NN algorithm for classifying the points on given graph.

Objective:

- The Basic Concepts of k-NN algorithm.
- Implementation logic of k-NN algorithm for classifying the points on given graph.

Theory:

Introduction

KNN is a non-parametric, lazy learning algorithm. Its purpose is to use a database in which the data points are separated into several classes to predict the classification of a new sample point.

When we say a technique is non-parametric, it means that it does not make any assumptions on the underlying data distribution. In other words, the model structure is determined from the data. Therefore, KNN could and probably should be one of the first choices for a classification study when there is little or no prior knowledge about the distribution data.

The KNN algorithm assumes that similar things exist in close proximity. In other words, similar things are near to each other.

KNN captures the idea of similarity (sometimes called distance, proximity, or closeness) with some mathematics we might have learned in our childhood— calculating the distance between points on a graph.

There are other ways of calculating distance, and one way might be preferable depending on the problem we are solving. However, the straight-line distance (also called the Euclidean distance) is a popular and familiar choice.

KNN Algorithm

We can implement a KNN model by following the below steps:

- Load the data
- Initialize the value of k
- For getting the predicted class, iterate from 1 to total number of training data points
    - Calculate the distance between test data and each row of training data. Here we will use Euclidean distance as our distance metric since it‘s the most popular method. The other metrics that can be used are Chebyshev, cosine, etc.
    - Sort the calculated distances in ascending order based on distance values
    - Get top k rows from the sorted array
    - Get the most frequent class of these rows
    - Return the predicted class

Choosing the right value for K

To select the K that‘s right for your data, we run the KNN algorithm several times with different values of K and choose the K that reduces the number of errors we encounter while maintaining the algorithm‘s ability to accurately make predictions when it‘s given data it hasn‘t seen before.

Here are some things to keep in mind:

- As we decrease the value of K to 1, our predictions become less stable. Just think for a minute, image K=1 and we have a query point surrounded by several reds and one green (I‘m thinking about the top left corner of the colored plot above), but the green is the single nearest neighbor. Reasonably, we would think the query point is most likely red, but because K=1, KNN incorrectly predicts that the query point is green.
- Inversely, as we increase the value of K, our predictions become more stable due to majority voting / averaging, and thus, more likely to make more accurate predictions (up to a certain point). Eventually, we begin to witness an increasing number of errors. It is at this point we know we have pushed the value of K too far.
- In cases where we are taking a majority vote (e.g. picking the mode in a classification problem) among labels, we usually make K an odd number to have a tiebreaker.

Advantages

- No assumptions about data—useful, for example, for nonlinear data
- Simple and easy to implement algorithm—to explain and understand/interpret
- High accuracy (relatively)—it is pretty high but not competitive in comparison to better supervised learning models
- Versatile—useful for classification or regression

Disadvantages

- Computationally expensive—because the algorithm stores all of the training data
- High memory requirement
- Stores all (or almost all) of the training data
- Prediction stage might be slow (with big N)
- Sensitive to irrelevant features and the scale of the data
- The algorithm gets significantly slower as the number of examples and/or predictors/independent variables increase.

Conclusion: We have studied the k-NN Classification algorithm and also implemented successfully.

Assignment No.4

Aim: To implement K-means Algorithm for clustering.

Objective:

- The Basic Concepts of K-means algorithm.
- Implementation logic of K-means algorithm.

Theory:

Introduction:

K-means clustering is one of the simplest and popular unsupervised machine learning algorithms.

Typically, unsupervised algorithms make inferences from datasets using only input vectors without referring to known, or labeled, outcomes.

A cluster refers to a collection of data points aggregated together because of certain similarities.

You‘ll define a target number k, which refers to the number of centroids you need in the dataset. A centroid is the imaginary or real location representing the center of the cluster.

Every data point is allocated to each of the clusters through reducing the in-cluster sum of squares.

In other words, the K-means algorithm identifies k number of centroids, and then allocates every data point to the nearest cluster, while keeping the centroids as small as possible.

The _means‘ in the K-means refers to averaging of the data; that is, finding the centroid.

To process the learning data, the K-means algorithm in data mining starts with a first group of randomly selected centroids, which are used as the beginning points for every cluster, and then performs iterative (repetitive) calculations to optimize the positions of the centroids

It halts creating and optimizing clusters when either:

The centroids have stabilized—there is no change in their values because the clustering has been successful.

The defined number of iterations has been achieved.

The procedure follows a simple and easy way to classify a given data set through a certain number of clusters (assume k clusters) fixed a priori.

The k-means algorithm takes the input parameter, k, and partitions a set of n objects into *k* clusters so that the resulting intra-cluster similarity is high but the inter-cluster similarity is low.

Cluster similarity is measured in regard to the mean value of the objects in a cluster, which can be viewed as the cluster‘s centroid or center of gravity.

The k-means algorithm proceed as follows

First, it randomly selects k of the objects, each of which initially represents a cluster mean or center.

For each of the remaining objects, an object is assigned to the cluster to which it is the most similar, based on the distance between the object and the cluster mean.

It then computes the new mean for each cluster. This process iterates until the criterion function converges.

Typically, the square-error criterion is used, defined as

$$E = \sum_{i=1}^{k} \sum_{p \in C_i} |p - m_i|^2,$$

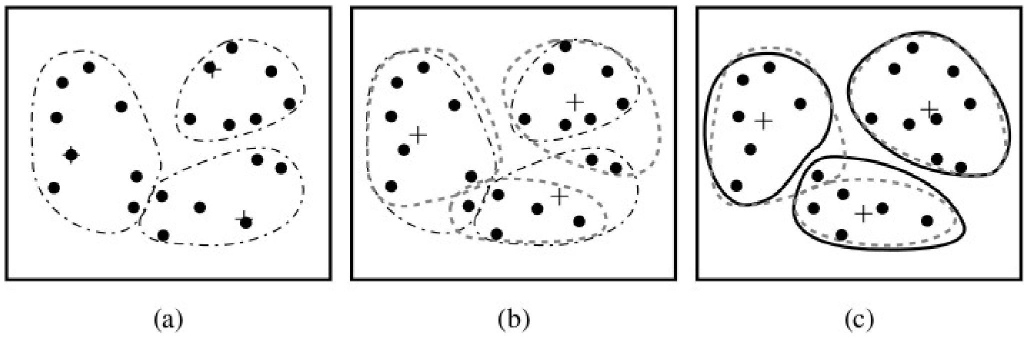
where *E* is the sum of the square error for all objects in the data set;

p is the point in space representing a given object;

mi is the mean of cluster *Ci* (both p and mi are multidimensional).

In other words, for each object in each cluster, the distance from the object to its cluster center is squared, and the distances are summed. This criterion tries to make the resulting k clusters as compact and as separate as possible.

The k-means procedure is summarized in following figure.



(a) (b) (c)

Above figure represent clustering of set of objects based on clustering methods. In figure, mean of each cluster marked by a ―+‖

K-means Algorithm

The k-means algorithm for partitioning, where is cluster center is represented by the mean value of the object in the cluster.

Input:

K: Number of clusters

D: Data Set containing n objects.

Output:

A set of K clusters

Algorithm:

1. Arbitrary choose k objects from D as the initial cluster centers
2. Repeat
3. (re)Assign each object to the cluster to which the object is most similar,
   Based on the mean value of the objects in the cluster;
4. Update the cluster means, i.e., calculate the mean value of the objects for each cluster,
5. Until no change

Advantages

- Easy to implement.
- An instance can change cluster (move to another cluster) when the centroids are re-computed.
- If variables are huge, then K-Means most of the times computationally faster than hierarchical clustering, if we keep k smalls.
- K-Means produce tighter clusters than hierarchical clustering, especially if the clusters are globular.

Disadvantages

- Difficult to predict the number of clusters. (K-Value)
- Initial seeds have a strong impact on the final results.
- The order of the data has an impact on the final results.
- Sensitive to scale: rescaling your datasets (normalization or standardization) will completely change results. While this itself is not bad, not realizing that you have to spend extra attention to scaling your data might be bad.

- Difficult to predict K-Value.
- With global cluster, it didn't work well.
- Different initial partitions can result in different final clusters.
- It does not work well with clusters (in the original data) of Different size and Different density

Conclusion: We have studied the k-means clustering algorithm and also implemented successfully.

ASSIGNMENT NO. 01

AIM: To implement a Simplified Data Encryption Standard (S-DES) algorithm.

OBJECTIVES:

- The Basic Concepts of S-DES.
- General structure of S-DES.
- Logical implementation of S-DES.

THEORY:

1. INTRODUCTION

Figure G.1 illustrates the overall structure of the Simplified DES, which we will refer to as S-DES. The S-DES encryption algorithm takes an 8-bit block of plaintext (example: 10111101) and a 10-bit key as input and produces an 8-bit block of cipher text as output. The S-DES decryption algorithm takes an 8-bit block of cipher text and the same 10-bit key used to produce that cipher text as input and produces the original 8-bit block of plaintext. The encryption algorithm involves five functions: an initial permutation (IP); a complex function labelled 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).

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 reduces 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).

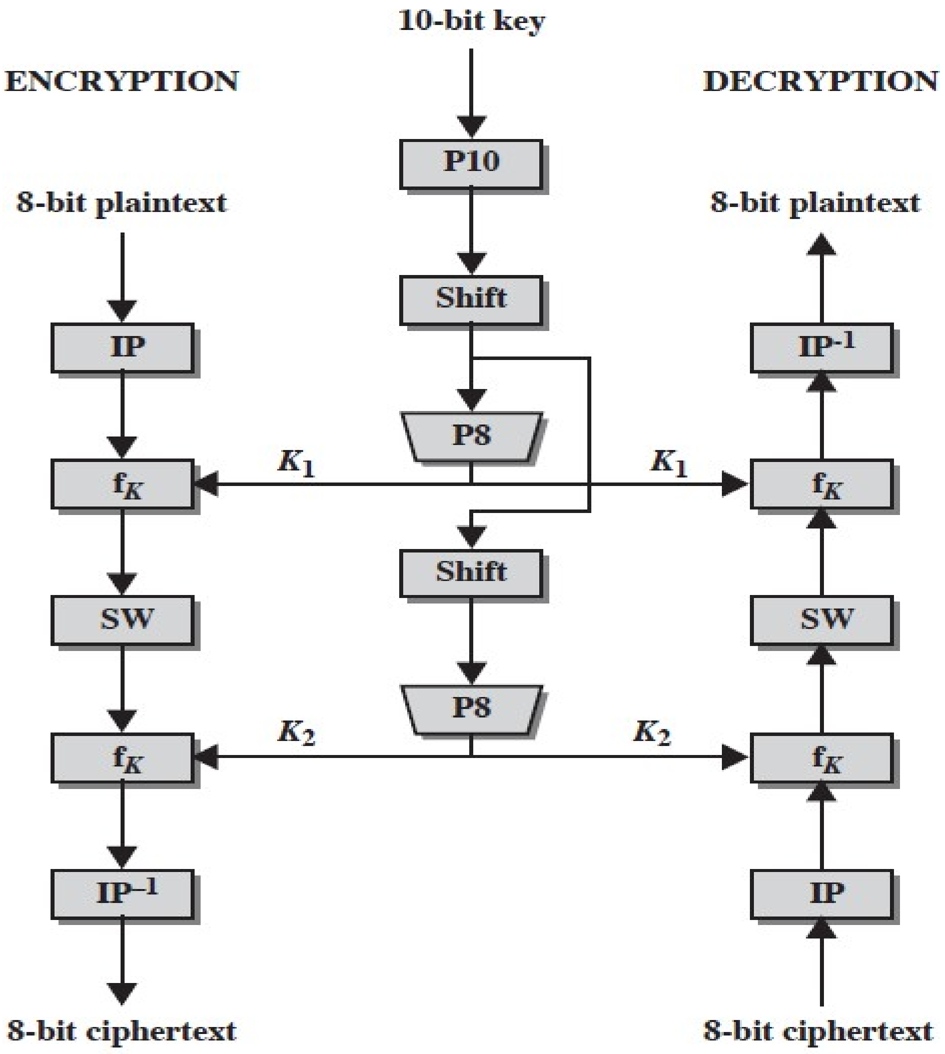


Figure G.1 Simplified DES Scheme

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

$$\mathrm{IP}^{-1} \circ \mathrm{f}_{K_2} \circ \mathrm{SW} \circ \mathrm{f}_{K_1} \circ \mathrm{IP}$$

which can also be written as:

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

Where

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

$$K_2 = \mathrm{P8}\Big(\mathrm{Shift}\big(\mathrm{Shift}(\mathrm{P10}(\mathrm{key}))\big)\Big)$$

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

$$\text{plaintext} = \mathrm{IP}^{-1}\Big(\mathrm{f}_{K_1}\Big(\mathrm{SW}\Big(\mathrm{f}_{K_2}\big(\mathrm{IP}(\text{ciphertext})\big)\Big)\Big)\Big)$$

2. S-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

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

as:

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).

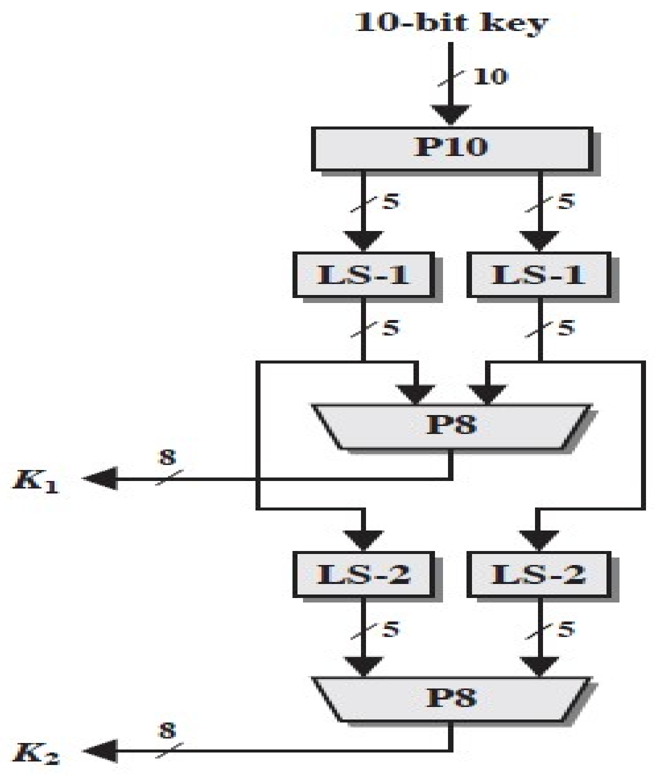


Figure G.2 Key Generation for Simplified DES

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 (0000111000) becomes (00100 00011). Finally, P8 is applied again to produce *K*2. In our example, the result is (01000011).

Figure G.3 shows the S-DES encryption algorithm in greater detail.

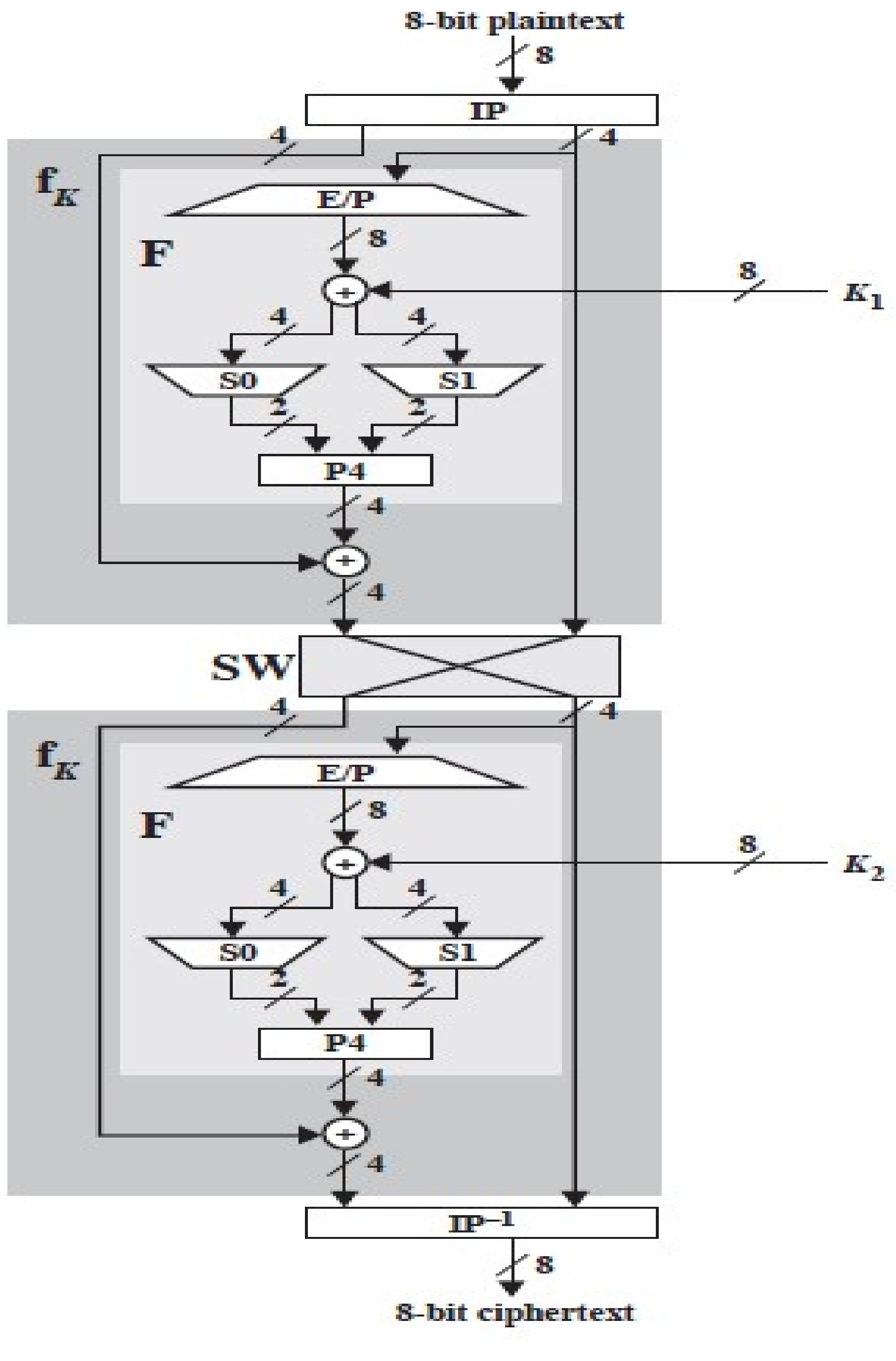


Figure G.3 Simplified DES Encryption Detail

3. FUNCTIONS

As it was mentioned, encryption involves the sequential application of five functions. We examine each of these.

a. 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 $IP^{-1}(IP(X)) = X$.

b.* 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:

E/P							
4	1	2	3	2	3	4	1

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

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

The 8-bit subkey $K1 = (k11, k12, k13, k14, k15, k16, k17, k18)$ is added to this value using exclusive-
OR:

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

Let us rename these 8 bits:

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

The first 4 bits (first row of the preceding matrix) are fed into the S-box S0 to produce a 2-bit output, and the remaining 4 bits (second row) are fed into S1 to produce another 2-bit output. These two boxes are defined as follows:

$$S0 = \begin{array}{c c} & \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \end{array} & \begin{bmatrix} 1 & 0 & 3 & 2 \\ 3 & 2 & 1 & 0 \\ 0 & 2 & 1 & 3 \\ 3 & 1 & 3 & 2 \end{bmatrix} \end{array} \qquad S1 = \begin{array}{c c} & \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \end{array} & \begin{bmatrix} 0 & 1 & 2 & 3 \\ 2 & 0 & 1 & 3 \\ 3 & 0 & 1 & 0 \\ 2 & 1 & 0 & 3 \end{bmatrix} \end{array}$$

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 Sbox. The entry in that row and column, in base 2, is the 2-bit output. For example, if $(p0,0p0,3) = (00)$ and $(p0,1p0,2) = (10)$, then the output is from row 0, column 2 of S0, which is 3, or (11) in binary. Similarly, $(p1,0p1,3)$ and $(p1,1p1,2)$ 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.

c. 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.

4. SOLVED EXAMPLE OF S-DES

(Note : Assignment for students)

ALGORITHM

CONCLUSION: Thus the implementation of S-DES algorithm was successfully conducted.

ASSIGNMENT NO. 02

AIM: To implement a Simplified Advanced Encryption Standard (S-AES) algorithm.

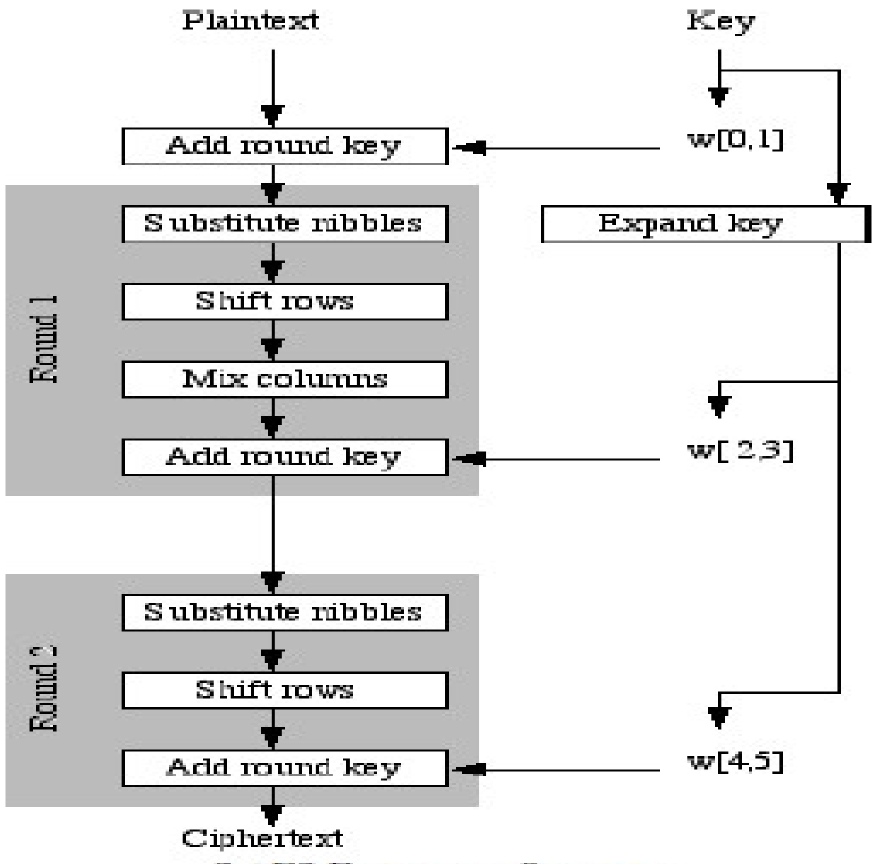
OBJECTIVES:

- The Basic Concepts of A S-AES
- General structure S-AES.
- Logical implementation of S-AES.

THEORY

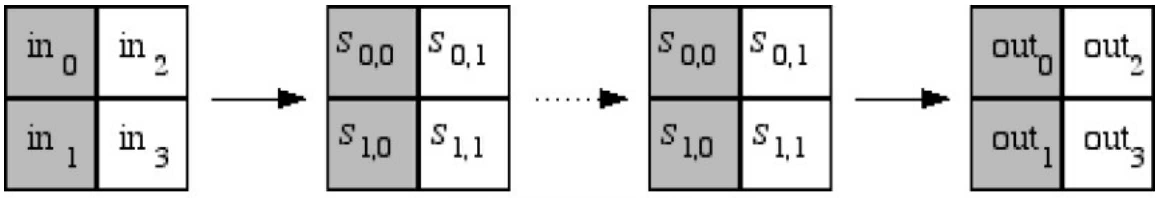
4. INTRODUCTION

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

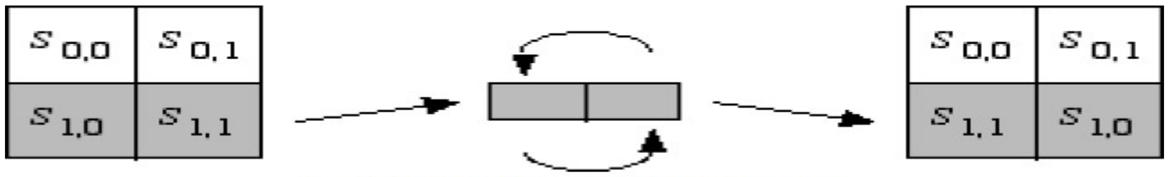
In the first stage of each encryption round, an S-box is used to translate each nibble into a new nibble. First we associate the nibble $b_0b_1b_2b_3$ with the polynomial $b_0x^3 + b_1x^2 + b_2x + b_3$. This polynomial is then inverted as an element of GF(16), with the –prime polynomial‖ used being.$x^4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

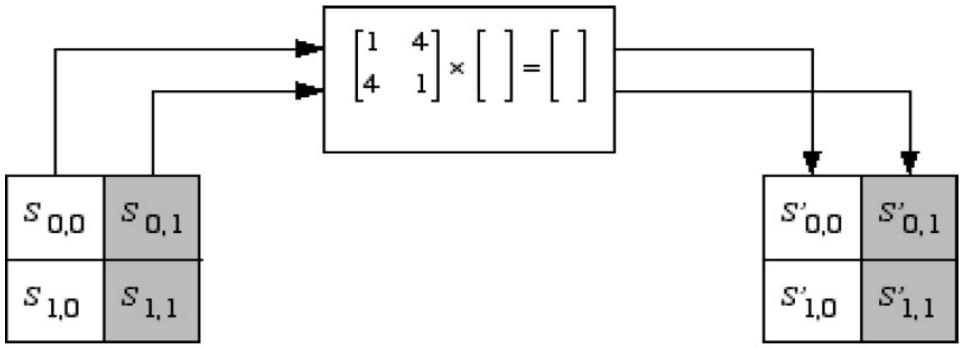
Mix Columns After shifting the rows, we mix the columns. Each column is multiplied by the Matrix

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

These operations are done in GF(16), so remember that the 1 corresponds to the polynomial 1 and the 4 corresponds to the polynomial x2. Thus this matrix could also be written as

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

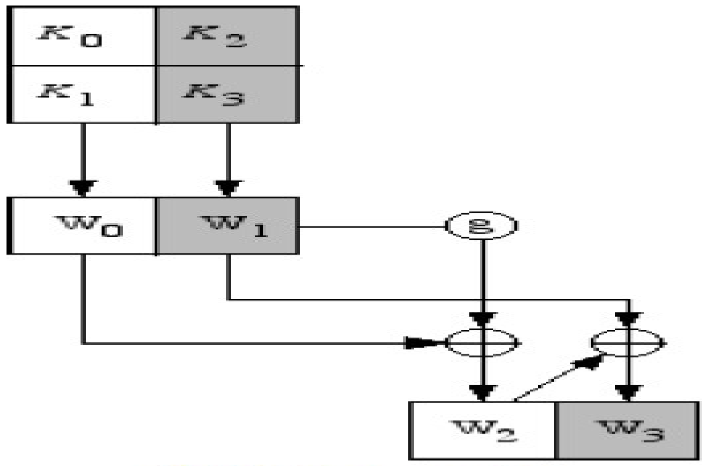
Don‘t forget to reduce modulo x4 + x + 1.
The mix column transformation is omitted in the last round, in order to simplify the decryption.



S-AES mix column transformation

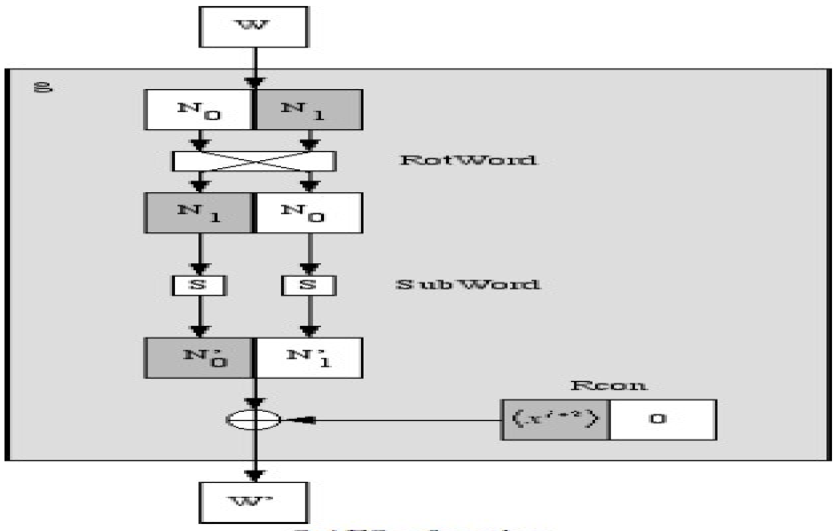
Add Round Key The last stage of each round of encryption is to add the round key. (In fact, this is also done before the first round.) Before the first round, the first two words (W0 and W1) of the expanded key are added. In the first round, W2 and W3 are added. In the last round, W4 and W5 are added. 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 W0 and W1 with W2 and W3, and replacing W2 and W3 with W4 and W5.



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.



S-AES g function

2. SOLVED EXAMPLE OF S-AES

(Note : Assignment for students)

ALGORITHM

CONCLUSION: Thus the implementation of S-AES algorithm was successfully conducted.

ASSIGNMENT NO. 03

AIM: To implement a Diffie-Hellman Key Exchange algorithm.

OBJECTIVES:

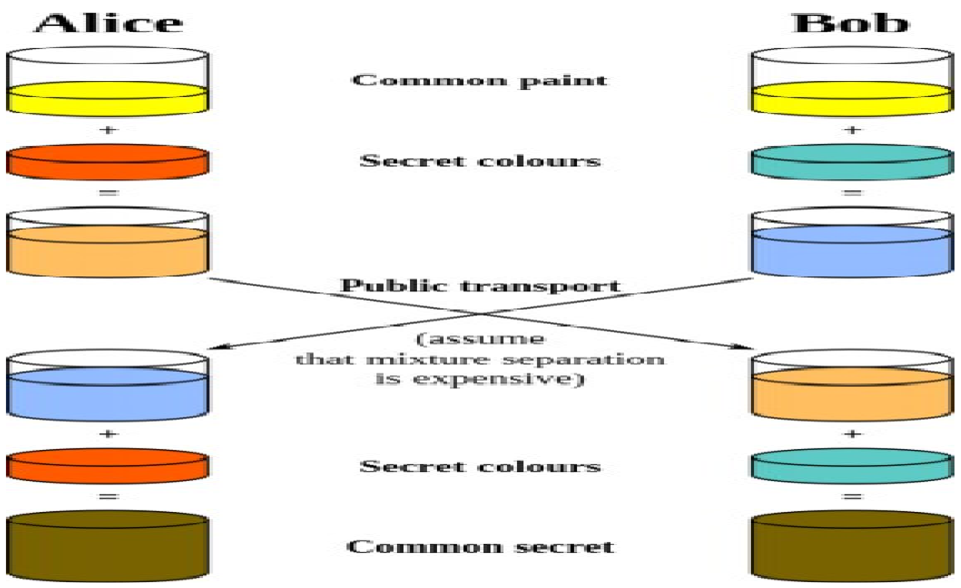
- The Basic Concepts of a Diffie-Hellman key exchange.
- General structure Diffie-Hellman key exchange.
- Logical implementation of Diffie-Hellman key exchange.

THEORY

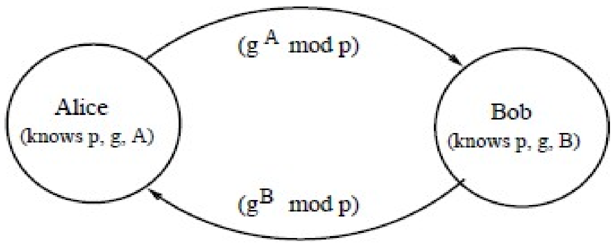
1. INTRODUCTION

Diffie–Hellman key exchange (**DH**) is a method of securely exchanging cryptographic keys over a public channel and was one of the first public-key protocols as originally conceptualized by Ralph Merkle and named after Whitfield Diffie and Martin Hellman DH is a mathematical algorithm that permits two PCs to produce an identical shared secret on both systems, despite the fact that those systems might never have communicated with one another. That shared secret can then be utilized to safely exchange a cryptographic encryption key. That key then encrypts traffic between the two systems. **Diffie-Hellman is not an encryption mechanism that is we don't commonly utilize DH to encrypt data. Rather, it is a strategy for secure exchange of the keys that encrypt data.**

The following diagram shows the general thought of the key exchange by utilizing colours rather than a large number. The procedure starts by having the two gatherings, Alice and Bob, agree on an arbitrary starting colour that does not should be kept secret; in this example the colour is yellow. Each of them chooses a secret colour - red and aqua respectively - that they keep to themselves. The vital piece of the procedure is that Alice and Bob now combine their secret colour with their commonly shared colour, bringing about orange and blue mixtures respectively, and then freely exchange the two mixed colours. At last, each of the two combine the colour they got from the partner with their own particular private colour. The outcome is a last colour mixture (brown) that is identical with the partner‘s colour mixture.



2. STEPS OF DEFFIE HELLMAN 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 ($g^a \mod p$).
3. Bob chooses a secret number b, and sends Alice ($g^b \mod p$).
4. Alice computes ($(g^b \mod p)^a \mod p$).
5. Bob computes ($(g^a \mod p)^b \mod p$).

Both Alice and Bob can use this number as their key. Notice that p and g need not be protected.

3. EXAMPLE OF DEFFIE HELLMAN KEY EXCHANGE

1. g = public (prime) base, known to Alice, Bob, and Eve.

   g = 5

2. p = public (prime) modulus, known to Alice, Bob, and Eve.

   p = 23

3. a = Alice‘s private key, known only to Alice.

   a = 6

4. b = Bob‘s private key known only to Bob.

   b = 15

5. A = Alice‘s public key, known to Alice, Bob, and Eve.

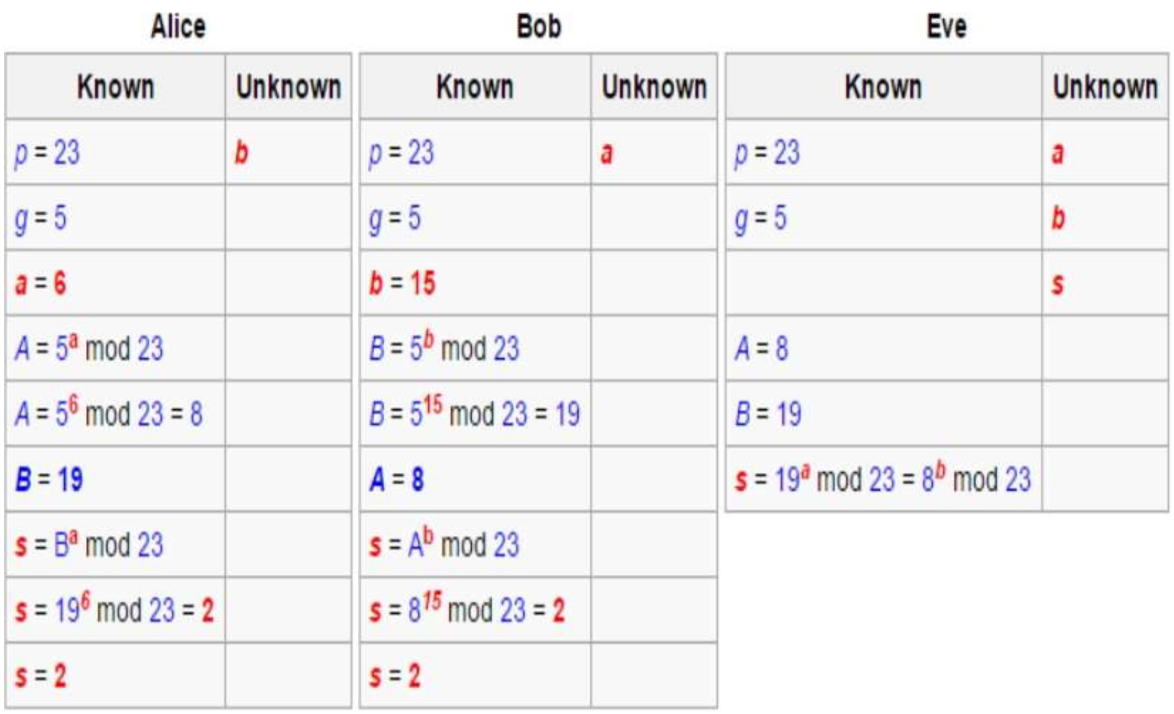
   A = g^a mod p = 8

6. B = Bob‘s public key, known to Alice, Bob, and Eve.

   B = g^b mod p = 19

7. Alice computes 19^6 mod 23 = 2
8. Bob computes 8^{15} mod 23 = 2
9. Then 2 is the shared secret.

   Above example is pictorially represented as follows :



Alice

Known	Unknown
$p = 23$	b
$g = 5$	
$a = 6$	
$A = 5^a \bmod 23$	
$A = 5^6 \bmod 23 = 8$	
$B = 19$	
$s = B^a \bmod 23$	
$s = 19^6 \bmod 23 = 2$	
$s = 2$	

Bob

Known	Unknown
$p = 23$	a
$g = 5$	
$b = 15$	
$B = 5^b \bmod 23$	
$B = 5^{15} \bmod 23 = 19$	
$A = 8$	
$s = A^b \bmod 23$	
$s = 8^{15} \bmod 23 = 2$	
$s = 2$	

Eve

Known	Unknown
$p = 23$	a
$g = 5$	b
	s
$A = 8$	
$B = 19$	
$s = 19^a \bmod 23 = 8^b \bmod 23$	

4. SECURITY ISSUE

Man-In-The-Middle Attack:

This algorithm has a noteworthy weakness as man - in - the - middle vulnerability. In this attack, a malicious third party, usually referred to as ―Eve‖ (for ―eavesdropper‖) recovers Alice‘s public key and sends her own public key to Bob. At the point when Bob transmits his public key, Eve interrupts on and substitutes the value with her own public key and after that sends it to Alice. At this point, Alice would have gone to an agreement on a common secret key with Eve rather than Bob and it is feasible for Eve to decrypt any messages conveyed by Alice or Bob, and after that read and perhaps alter them before the re-encryption with the suitable key and transmitting them to alternate party.

This weakness is available on the grounds that Diffie Hellman key exchange does not authenticate the members. Possible solutions include the use of digital signatures and other protocol variants.

5. ADVANTAGES

1. The security factors with respect to the fact that solving the discrete logarithm is very challenging.
2. That the shared key (i.e. the secret) is never itself transmitted over the channel.

6. DISADVANTAGES

1. The fact that there are expensive exponential operations involved, and the algorithm cannot be used to encrypt messages - it can be used for establishing a secret key only.
2. There is also a lack of authentication.
3. There is no identity of the parties involved in the exchange.
4. It is easily susceptible to man-in-the-middle attacks. A third party C, can exchange keys with both A and B, and can listen to the communication between A and B.
5. The algorithm is computationally intensive. Each multiplication varies as the square of n, which must be very large. The number of multiplications required by the exponentiation increases with increasing values of the exponent, x or y in this case.
6. The computational nature of the algorithm could be used in a denial of-service attack very easily.

ALGORITHM

CONCLUSION: Thus the implementation of Diffie-Hellman key exchange algorithm was successfully conducted.

ASSIGNMENT NO. 04

AIM: To implement a RSA algorithm.

OBJECTIVES: Students will learn:

- The Basic Concepts of RSA.
- General structure of RSA.
- Logical implementation of RSA.

THEORY:

1. INTRODUCTION

RSA (**Rivest–Shamir–Adleman**) is one of the first public-key cryptosystems and is widely used for secure data transmission. It is an asymmetric cryptographic algorithm. Asymmetric means that there are two different keys. This is also called public key cryptography, because one of them can be given to everyone. In such a cryptosystem, the encryption key is public and it is different from the decryption key which is kept secret (private). In RSA, this asymmetry is based on the practical difficulty of the factorization of the product of two large prime numbers, the "factoring problem". The acronym RSA is made of the initial letters of the surnames of Ron Rivest, Adi Shamir, and Leonard Adleman, who first publicly described the algorithm in 1978.

A user of RSA creates and then publishes a public key based on two large prime numbers, along with an auxiliary value. The prime numbers must be kept secret. Anyone can use the public key to encrypt a message, but with currently published methods, and if the public key is large enough, only someone with knowledge of the prime numbers can decode the message feasibly. Breaking RSA encryption is known as the RSA problem.

RSA is a relatively slow algorithm, and because of this, it is less commonly used to directly encrypt user data. More often, RSA passes encrypted shared keys for symmetric key cryptography which in turn can perform bulk encryption-decryption operations at much higher speed.

2. OPERATIONS

The RSA algorithm involves four steps: **key generation, key distribution, encryption and decryption.**

A basic principle behind RSA is the observation that it is practical to find three very large positive integers *e*, *d* and *n* such that with modular exponentiation for all integers *m* (with $0 \leq m < n$):

$$(m^e)^d = m \pmod n$$

and that even knowing *e* and *n* or even *m* it can be extremely difficult to find *d*.

In addition, for some operations it is convenient that the order of the two exponentiations can be changed and that this relation also implies:

$$(m^d)^e = m \pmod n$$

RSA involves a *public key* and a *private key.* The public key can be known by everyone, and it is used for encrypting messages. The intention is that messages encrypted with the public key can only be decrypted in a reasonable amount of time by using the private key. The public key is represented by the integers n and e; and, the private key, by the integer d(although n is also used during the decryption process. Thus, it might be considered to be a part of the private key, too). m represents the message.

a. Key generation

The keys for the RSA algorithm are generated the following way:

1. Choose two distinct prime numbers p and q.

   For security purposes, the integers p and q should be chosen at random, and should be similar in magnitude but differ in length by a few digits to make factoring harder. Prime integers can be efficiently found using a primarily test.
2. Compute $n = pq$.

   n is used as the modulus for both the public and private keys. Its length, usually expressed in bits, is the key length.

3. Compute $\lambda(n) = \mathrm{lcm}(\lambda(p), \lambda(q)) = \mathrm{lcm}(p - 1, q - 1)$, where λ is Carmichael's totient function. This value is kept private.
4. Choose an integer e such that $1 < e < \lambda(n)$ and $\gcd(e, \lambda(n)) = 1$; i.e., e and $\lambda(n)$ are co-prime.
5. Determine d as $d \equiv e^{-1} \pmod{\lambda(n)}$; i.e., d is the modular multiplicative inverse of e modulo $\lambda(n)$.
   - This means: solve for d the equation $d\square e \equiv 1 \pmod{\lambda(n)}$.
   - e having a short bit-length and small Hamming weight results in more efficient encryption – most commonly $e = 2^{16} + 1 = 65{,}537$. However, much smaller values of e (such as 3) have been shown to be less secure in some settings.
   - e is released as the public key exponent.
   - d is kept as the private key exponent.

The *public key* consists of the modulus n and the public (or encryption) exponent e. The *private key* consists of the private (or decryption) exponent d, which must be kept secret. p, q, and $\lambda(n)$ must also be kept secret because they can be used to calculate d.

b. Key distribution

Suppose that Bob wants to send information to Alice. If they decide to use RSA, Bob must know Alice's public key to encrypt the message and Alice must use her private key to decrypt the message. To enable Bob to send his encrypted messages, Alice transmits her public key (n, e) to Bob via a reliable, but not necessarily secret, route. Alice's private key (d) is never distributed.

c. Encryption

After Bob obtains Alice's public key, he can send a message M to Alice. To do it, he first turns M (strictly speaking, the un-padded plaintext) into an integer m (strictly speaking, the padded plaintext), such that $0 \leq m < n$ by using an agreed-upon reversible protocol known as a padding scheme. He then computes the cipher text c, using Alice's public key e, corresponding to

$$c \equiv m^e \pmod{n}$$

Bob then transmits c to Alice.

d. Decryption

Alice can recover m from c by using her private key exponent d by computing

$$m \equiv c^d \pmod{n}$$

Given m, she can recover the original message M by reversing the padding scheme.

3. STEPS OF RSA ALGORITHM

STEP-1: Select two co-prime numbers as p and q.

STEP-2: Compute n as the product of p and q.

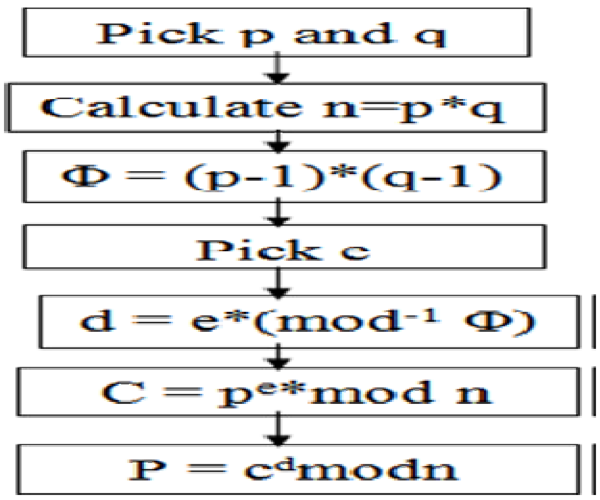
STEP-3: Compute (p-1)*(q-1) and store it in z.

STEP-4: Select a random prime number e that is less than that of z.

STEP-5: Compute the private key, d as e * mod^{-1} (z).

STEP-6: The cipher text is computed as $message^{e}$ * mod n.

STEP-7: Decryption is done as $cipher^{d}$ * mod n.



4. EXAMPLE OF RSA ALGORITHM

1. Select primes: p = 17 ; q = 11

2. Calculate n = pq = 17 x 11 = 187

3. Calculate ø(n) = (p–1)(q-1) = 16x10 = 160

4. Select e: GCD(e,160) = 1 ; choose e = 7

5. Derive d: de = 1 mod 160 and d < 160

Get d = 23 since 23x7 = 161 = 10x160+1

6. Publish public key: PU = {7,187}

7. Keep private key secret: PR = {23,187}

RSA Encryption and Decryption :

• Sample RSA encryption/decryption is:

• given message M = 88 (nb 88 < 187)

• Encryption:

$C = 88^7 \bmod 187 = 11$

• Decryption:

$M = 11^{23} \bmod 187 = 88$

ALGORITHM

CONCLUSION: Thus the implementation of RSA algorithm was successfully conducted.