CHAPTER 1

1 INTRODUCTION: Within last decade there has been a vast increase in the accumulation and communication of digital computer data in all public, private and most important defense sectors. Much of this data has a significant value. Sensitive data concerning the organization is processed by computer systems. Thus it becomes vulnerable. So either directly or indirectly, it requires protection.

The rapid growth of computer data banks increases threats to personal and organizational privacy. Since this information is often accessible from remote terminals, there is a threat of easy, unchecked and unauthorized access to the information from any place in the data communication systems.

As introductory remarks ENCRYPTION may be used for data security applications. With improper use and implementation data encryption may provide only an illusion of security. With inadequate understanding of encryption applications, data encryption could deter the utilization of other needed protecting techniques. However with proper management controls, adequate understanding and careful implementation, data encryption will not only help in protecting data communications but can provide a myriad of specific data processing applications.

1.1 Objectives: This application is developed after keeping in view the organizational compulsion of secrecy in defense applications. Nevertheless this application can also be used by other organizations, which are very particular about safekeeping of their critical data. Moreover, computer networks have become part and parcel of almost every large and small organization. Thus communication within the network and out side is never safe, if no special measures are taken for data protection. Military has also become one of the biggest users of networks in present world. That is mainly due to increase in the role of military from local defenses to global deployment.

Pakistan is a very active member of UN. By virtue of its importance in the region particularly and in the globe generally, Pakistan military forces are being deployed all around the world for different type and natures of duties. Consequently to communicate with the forces deployed outside the Pakistan, a very secure application is needed to protect the data of critical nature.

1.2 Technical achievements: Once the syndicate undertook the project, it

was merely an idea. To design it, was really a gigantic task, because it was to be started from nowhere. There was a misunderstanding that this project was previously done, but never succeeded. While tracing that project we found out that it was misleading rather than being of some help.

We started it again from "ZERO". And by the grace of Almighty Allah, we were able to make following achievements (only introduction is given here).

- We were able to develop a truly user-friendly application.
- It is a totally platform independent application.
- User can select any data file (text) of any size from his system.
- User can encrypt the file with or without use of enhanced encryption technique of "**DATA ENCRYPTION STANDARD**".
- User can select image of his own choice but of .BMP file extension.
- Our application is capable of hiding the data, encrypted or unencrypted, in an image without distortion.
- Application is capable of decrypting the file hidden in an image.
- The application can be placed in highly secure rating because of the "KEY" it uses for encrypting and decrypting data.
- To decrypt the file user must use the same key used for encryption.
- The application guides the user at run time if he goes astray. Which enhances the relation between the user and the application.
- Besides these we learnt a lot during the development of the project in the form of java language, cryptography, how to tackle and overcome the problems encountered.

1.3 Software Processes: Like any other application this application also employs a lot of different software processes. All of them are listed and introduced as follow.

1.3.1 Encryption: There is an independent module for encryption. This module requires a text file, which is to be encrypted. Basically it uses a 64-bit key to run its encryption process. Once the key is fed ion it precedes on to encryption. This module works on 64-bit key and 64 bit block for encryption.

1.3.2 Decryption: The decryption module works on the same footing as encryption module. But if the key for decryption is other than the key from encryption then the resultant document will be a garbage and unintelligible document.

1.3.3 Image Processing: Image processing module reads in the user given image. It then reads its pixels and then hides the encrypted text in it. The same module retrieves the hidden text once decryption is done.

Chapter 2

JAVA PROGRAMMING LANGUAGE:

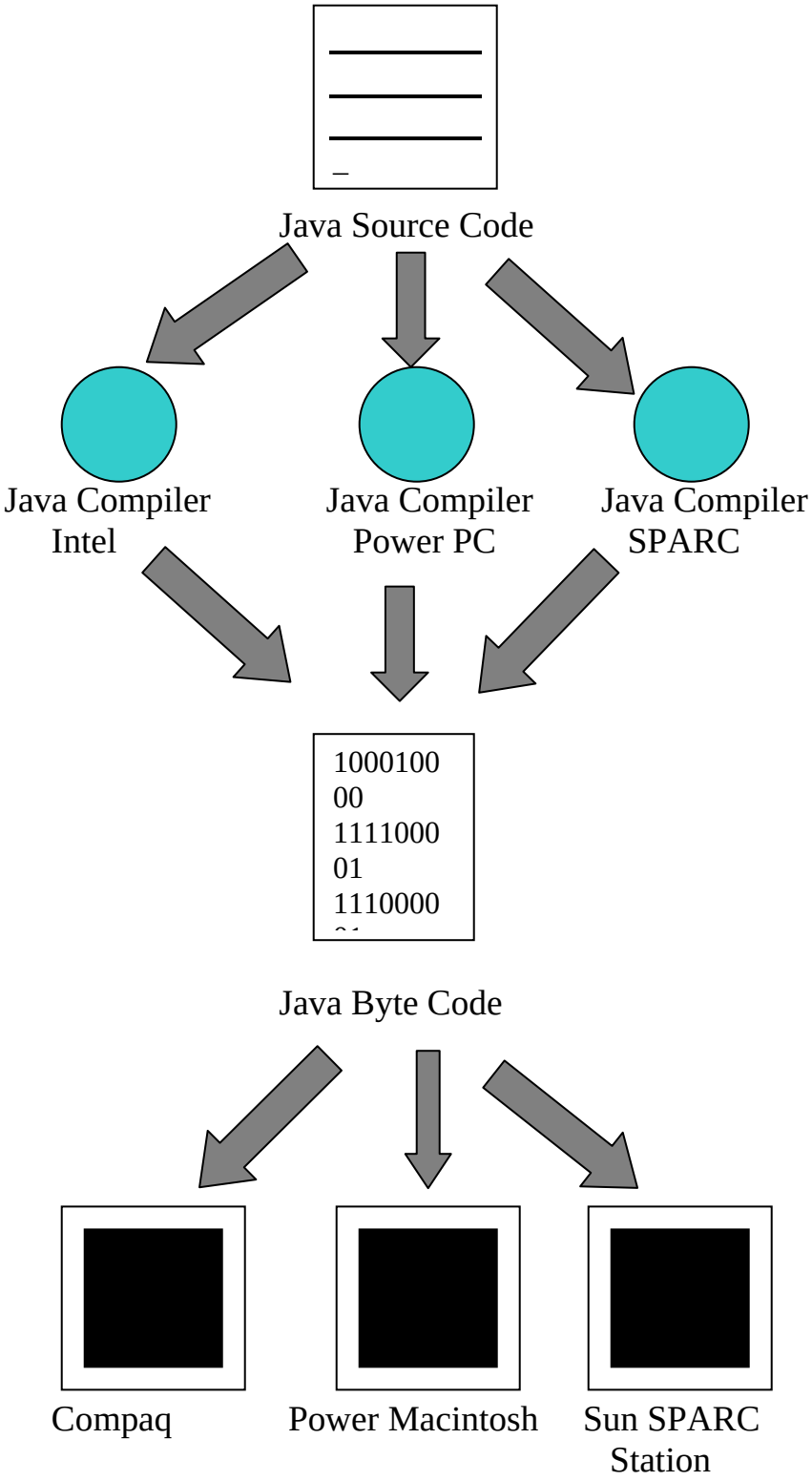
2.1 Introduction: In 1995, Sun Microsystems introduced the Java programming language to the world. Although this language derives heavily from C++, it has many special features that have made it a huge success with programmers at all levels. For example the portable nature of Java program makes it ideal for developing applets to be run on the World Wide Web. However it is easy, and often as advantageous to develop applications ion Java that run completely outside of a browser.

The java programming language provides an excellent opportunity that produces appealing and sophisticated standalone applications. Additionally, the object-oriented nature of Java allows programmers to develop the code that easily can be reused in other applications. Although there are many different definitions of the java programming language today, most of them are similar to the following:

> "Java is a simple, object-oriented, secure, interpreted, platform independent, portable, multithreaded language ".

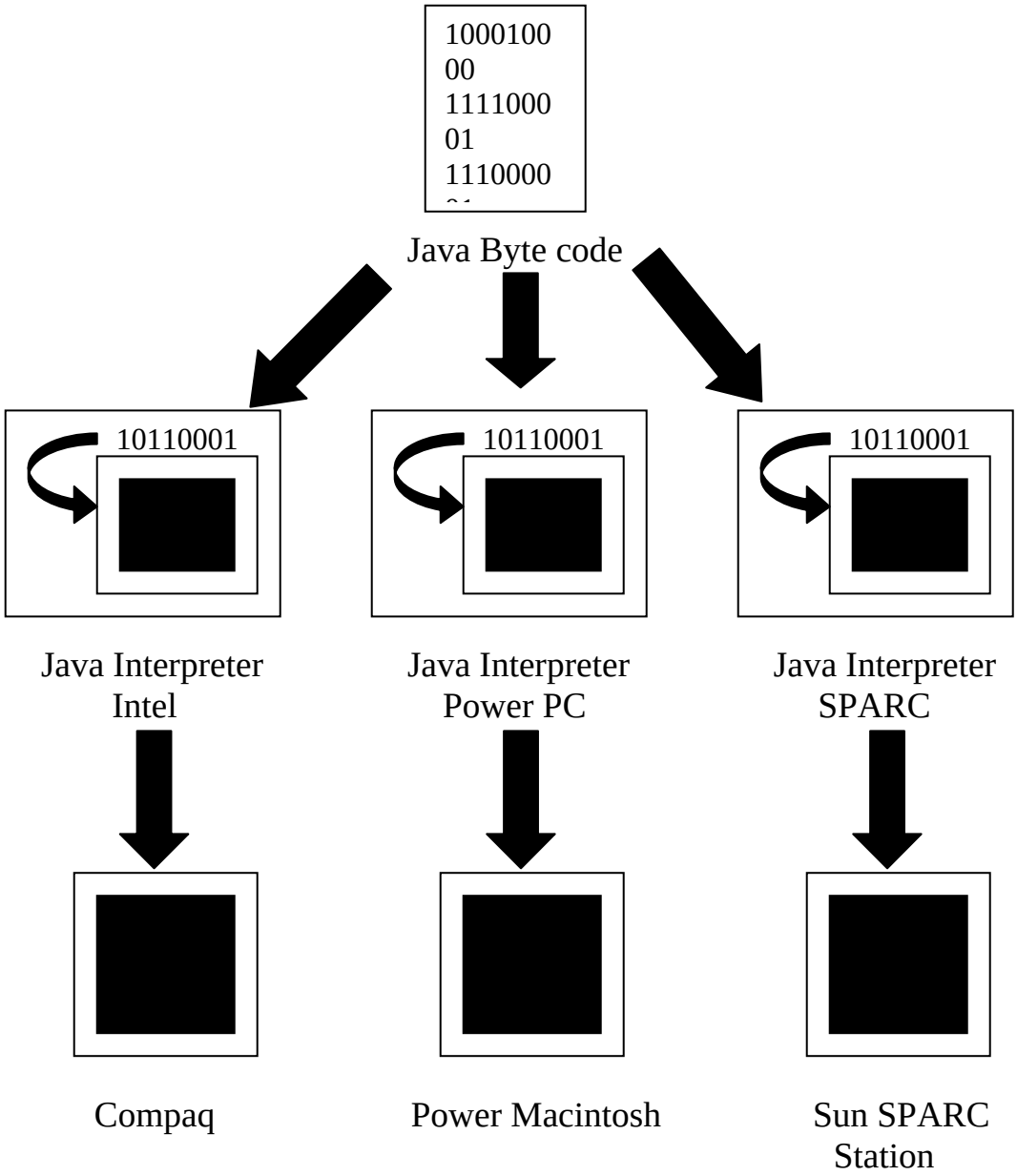
2.2 Java is Platform-independent: Java program unlike those in other programming languages are platform-independent. This allows the java applets and applications to be run on varying platforms without recompiling the source code. This feature is realized because Java is an interpreted language. This means that the byte code of the Java application can be executed independently of the platform the user is running without any changes. The disadvantage of the byte code is execution speed.

2.2.1 Java byte code and java Interpreters: When a java source code is compiled to create an application and applet, the resulting executable, can run on a variety of machines-regardless of the platform used for development. This happens because the java compiler generates a byte code file, which looks the same no matter what kind of computer you use as shown in the figure.



Identical byte code produced by the java compiler that can run on different machines

A java byte code interpreter is specific to the machine it runs on. For example, the java interpreter used on a power Macintosh knows how to interpret a byte code file, so it runs properly on a computer driven by a power PC Chip. A different java interpreter, such as the one used on the computer running Windows 95, knows how to interpret the same byte code file, so it executes on a computer with an Intel microprocessor as shown in the figure.



Different machines can run the same byte code, but they require different interpreters

2.3 Java Applications: Java can be used to create full-scale applications as well as applets. After the source code for the java application is written and compiled. The end result is a single executable file that can be run on all of the popular computer systems in the market. The interpreter is itself a stand-alone software application. Java code can run on any machine –provided that the machine has a java interpreter. At the time of this writing, the application interpreter is available for computers running Windows, Windows NT and Solaris 2.3 or higher version.

2.4 Reasons for choosing java as developing tool: Following are the characteristic which made us to use java as a programming language for this project:

2.4.1 Simple: As it is already mentioned that the java has got its origin from C and C++. So transforming from C++ to Java was not very difficult for us. But while java is quite similar to C and C++, it is actually much simpler. Memory management probably the most bugging area of programming has been greatly simplified by Java's elimination of pointers. Phrases like "Dangling pointers" and "Memory leaks" are not part of Java terminology.

2.4.2 Object-oriented: an object-oriented language forces programmers to focus not only on data but also how data be manipulated. The chief advantage of an object-oriented language is said to be its "code reusability". Sun Microsystems and Microsoft have fostered code reusability by developing a huge set of reusable java classes and by making these classes freely available to java programmers. Moreover one can find thousands of reusable classes left on the Internet for free use by the programmers.

2.4.3 Distributed: java is the language of the Web because it fully supports applications that were developed to execute over a network. So in order to have a touch of Internet programming we opted for java as our choice for software development.

2.4.4 Interpreted: as it is already mentioned that java is an interpreted language. So we wanted to make our application to be able to run on any machine because it will be best utilized only when it will be

distributed or installed on number of computers irrespective of the operating system installed on those computers.

2.4.5 Exception handling: the java programming language provides a construct for handling certain types of errors. This is known as exception and exception handling. In our application we have defined our own classes to throw exceptions if a certain error condition is encountered.

2.4.6 Multithreading: one of the major advantages of java programming language is that it has built in support for multithreading. In other words, applications can perform multiple tasks simultaneously. This feature also compensates for the some of the relatives slowness introduced by java's interpretive nature.

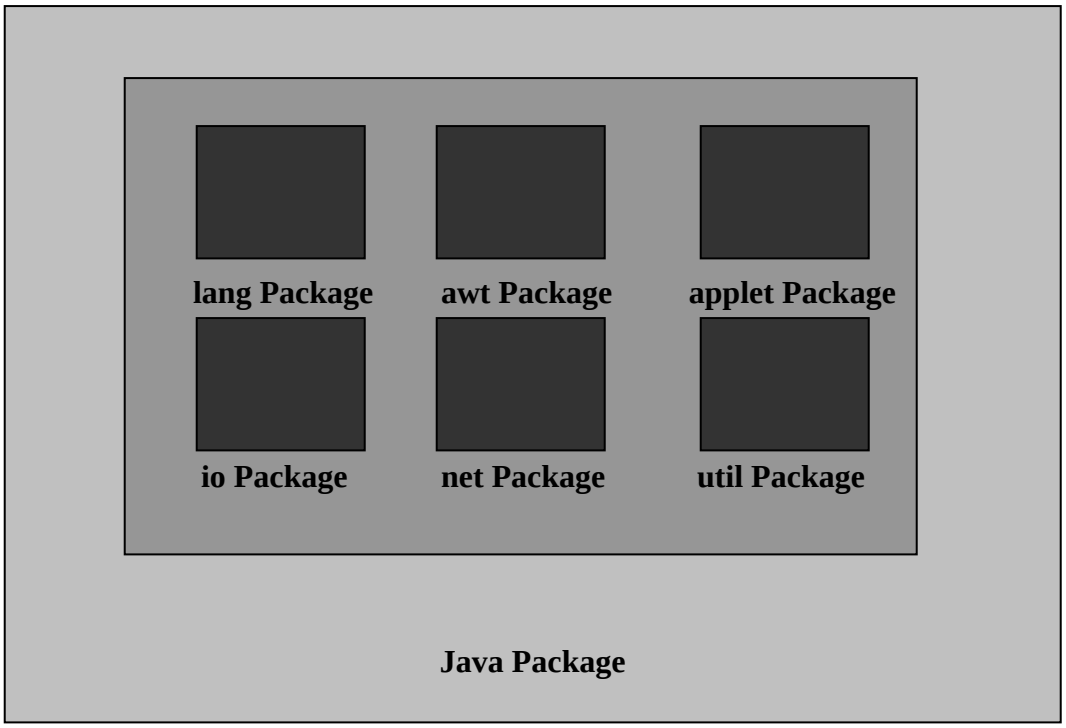
2.4.7 Garbage collection: is one of the unique features of java. The part of temporary memory, which is not referred to by any active part of program, is automatically reclaimed

2.4.8 Java packages: A package is the way of grouping together several logically related classes. A package holds the compiled code for several classes. To reference a single class in a package, both the package hierarchy and the class separated by periods have to be mentioned. unlike a C library, which holds the object code of compiled functions, a java package holds the object code of compiled classes. The under mentioned table gives a short overview of purpose of each of the six packages found in the java package.

Java's Six Packages

Java Packages	Purpose
Java.lang	Holds the essential java classes-classes that define the java language.
Java.io	Holds the classes used when working with data input and output.
Java.awt	Holds the classes used to give your applications a graphical user interface AWT stands for Abstract windowing toolkit.

Java.applet	Holds the classes necessary to create an applet.
Java.net	Holds the classes used for communicating with a network.
Java.util	Holds the classes used for utility functions

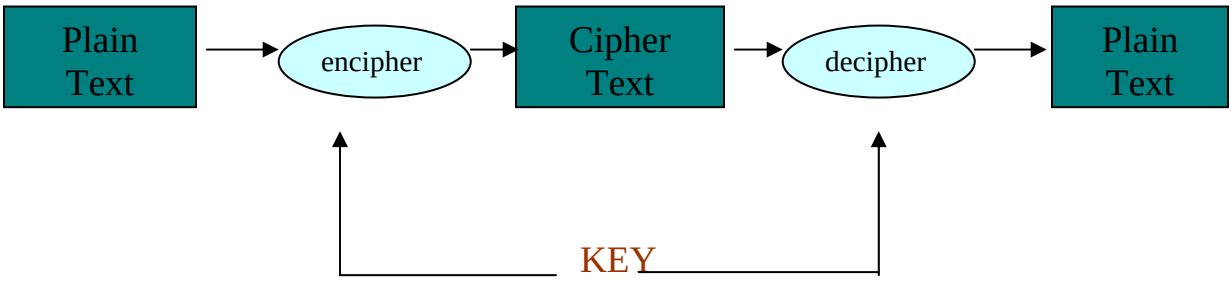


The purpose of java package is to contain other packages

Chapter 3

CRYPTOGRAPHY

3.1 Introduction: Classical cryptography provided secrecy for information sent over channels where eavesdropping and message interception was possible. The sender selected a cipher and encryption key, and either gave it directly to the receiver or else sent it indirectly over a slow but secure channel (typically a trusted courier). Messages and replies were transmitted over the insecure channel in cipher text as shown in the figure.



Modern cryptography protects data transmitted over high-speed electronic lines or stored in computer systems. There are two principal objectives: **secrecy** (for privacy) to prevent the unauthorized disclosure of data, and **authenticity** or **integrity** to prevent the unauthorized modification of the data.

Information transmitted over electronic lines is vulnerable to passive wiretapping, which threatens secrecy, and to active wiretapping which threatens authenticity. Passive wiretapping refers to the interception of message usually without detection. Protection against disclosure of message contents is provided by enciphering transformations and by the cryptographic techniques. Active wiretapping refers to the deliberate

modifications made to the message stream. This can be done for the purpose of making arbitrary changes to a message, or replacing the data in a message with replays of data from earlier messages. Encryption protects against message modification and injection of false messages by making it infeasible for an opponent to create cipher text that deciphers into meaningful plain text.

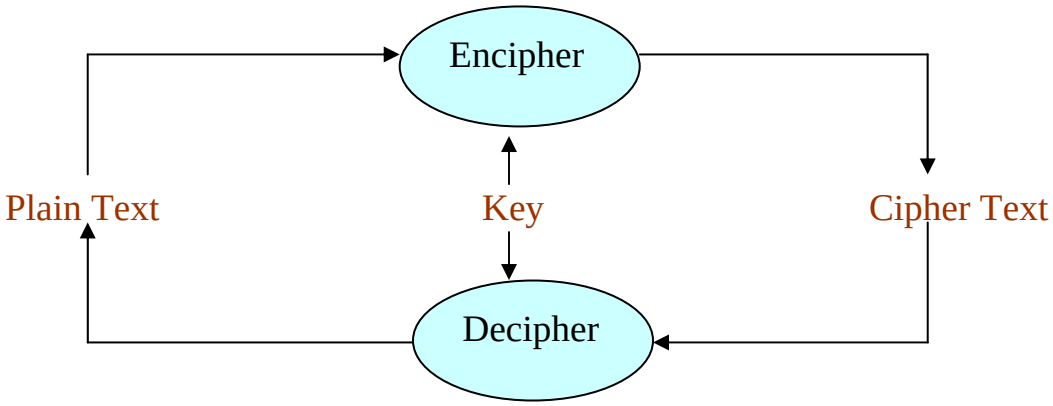
3.2 Cryptographic terms:

3.2.1 Cryptography: is the science and study of secret writing.

3.2.2 Cipher: is a secret method of writing, whereby plaintext (or **clear text**) is transformed into **cipher text (**sometimes called as **cryptogram)**.

3.2.3 Encryption: The process of transforming plain text into cipher text is called **encipherment** or **encryption**.

3.2.4 Decryption: The reverse process of transforming cipher text into plain text is called **decipherment or decryption**.



3.2.5 Transposition Ciphers: In **transposition ciphers** bits or characters in the data are rearranged.

3.2.6 Substitution Ciphers: in **substitution ciphers** bits, characters or blocks of characters are replaced. In computer applications, transposition is usually combined with substitution as it is done in this application in case of **DES**.

3.3 Threats to Data Stored in Computer System: Various threats to data stored in computer systems are as under:

3.3.1 Overwriting: As name refers, it means completely changing the correct data with an incorrect data.

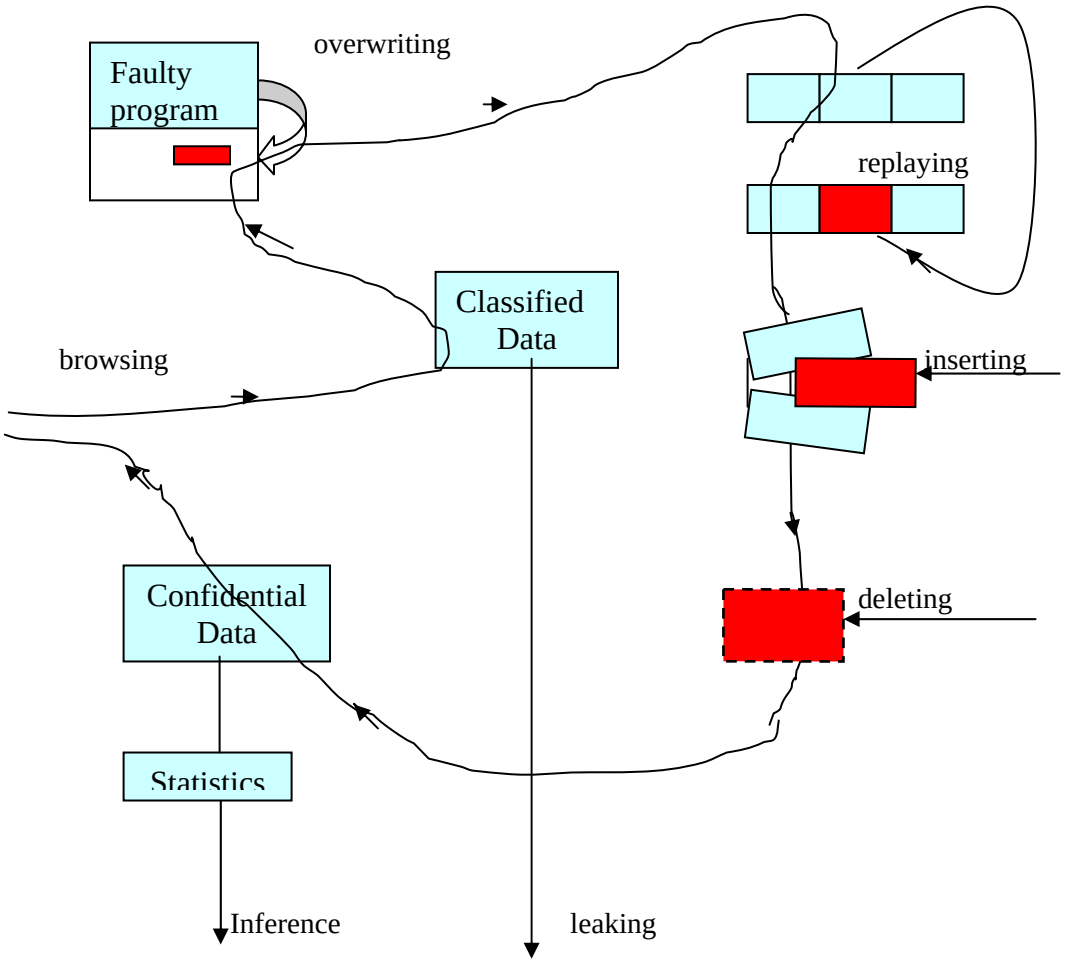
3.3.2 **Modifying:** As name implies, the intruder may modify the existing data with some changes thus making the data incorrect. The difference between overwriting and modifying is that in case of overwriting, the data is completely replaced by incorrect data by the intruder. While in case of modification, Part of the data is replaced with the incorrect data.

3.3.3 **Replaying:** In this case, the current data is replaced by an old data. This type of threat is very difficult to be detected as it makes use of the correct data but of older version.

3.3.4 **Inserting:** In this case, intruder inserts some data in the actual data. This is done with the intention to change the context of the actual data.

3.3.5 **Deleting:** In this case, some important information is deleted from the actual data. Thus making the actual data incorrect or unreadable.

3.3.6 **Leaking:** Here the information is leaked to some user who is not entitled to have that information. No modification is made to the data. Thus making it almost impossible to detect the unauthorized access to the data.



3.4 Where Should Data Encryption Be Used? Cryptography (encryption) has historically been used to protect sensitive information darn~ communication. It can be used for protecting computer data transmitted between terminals and computers or between computers. Data is encrypted before transmission and decrypted after it is received. The algorithm used to decrypt the received cipher must be the inverse of the algorithm used to encrypt the transmitted data. In general, a device used to transmit and receive data would contain algorithms for both encryption and decryption.

Encryption can be used between data processing machines and data storage devices such as magnetic tape and magnetic disk. In this application, the data is encrypted before it is written on the storage device and decrypted before it is subsequently read. Data is stored in its cipher form and transformed to plaintext only when it is to be processed within the computer.

Encryption can be used to authenticate the identities of users, terminals, and computers of a data processing system. Passwords have historically been used to differentiate between friend and foe during times of war. Knowledge of the secret password was accepted as authenticating the identity of friends. Unique identification was not necessary and the password was changed for each mission. The DES uses a key, similar to a password, which must be supplied to each group of users of the algorithm. Having the correct key authenticates an individual to a data processing system.

In a similar manner a terminal or a computer may be authenticated as an authorized device of a data processing system. Supplying the correct key to a DES device when requested by the authorization system can authenticate a terminal associated with the device. This authorization system may be a special program or a special computer system that has been established to control access to the resources and data of the overall system. The authorization system must be initialized with the identities and the authentication keys of all authorized users and devices of the system. This system will issue a challenge for proper identification whenever a device or individual wishes to access the system. Similar challenge/response password systems are currently in use for computer user authentication. When combined with data encryption technology, authorization systems can authenticate the claimed identities of users and devices without

compromising the passwords or keys by transmitting them through the system.

3.4 **When Should Data Encryption Be Used?** Data encryption should be used whenever it is the most cost effective method available to protect the confidentiality or integrity of the data. Confidentiality refers to the accidental or intentional disclosure of data to an unauthorized individual. Integrity refers to data that has not been exposed to accidental or malicious alteration or destruction. Encryption of data prevents unauthorized recipients of the cipher from interpreting its meaning. Encryption can also prevent unauthorized individuals from manipulating the cipher in such a way that the original data is changed in a predetermined manner. To be effective, encryption must cost less than the expected loss (risk) if the protection were not provided. Computation or estimation of costs and risks and the decision to employ cryptographic protection are management functions of the authority responsible for the data.

3.5 DES:

3.5.1 **Overview:** DES stands for **Data Encryption Standard**. It's the first standard cipher business world had and is still the most commonly used. IBM developed the encryption algorithm for DES. Des enciphers 64-bit blocks of data with a 56-bit key. As it is a standard, any system using DES could talk to any other system using it (but they always had to find a way to agree on the key to use). No one has published a system for cracking DES, except the brute force method of trying all keys until one works.

3.5.2 **Requirements of DES:** An encryption algorithm must satisfy the following requirements in order to be acceptable as a Federal standard:

It must provide a high level of security.

- It must be completely specified and easy to understand.
- The security provided by the algorithm must not be based upon the secrecy of the algorithm.
- It must be available to all users and suppliers.
- It must be adaptable for use in diverse applications.
- It must be economical to implement in electronic devices and be efficient to use.
- It must be amenable to validation.

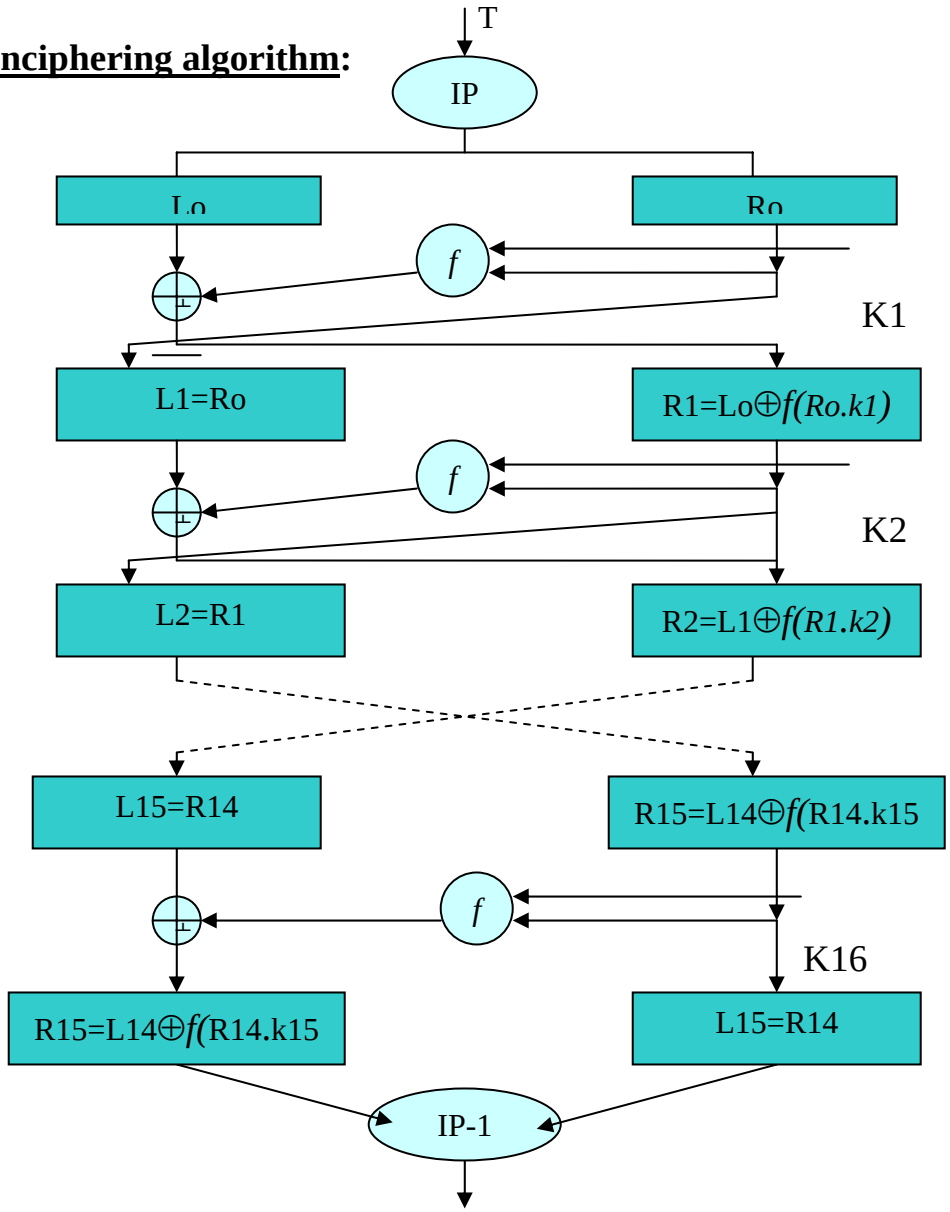
- It must be exportable.

The DES satisfies all these requirements.

3.5.3 **Strength of DES:**

- DES with a key of 56-bits means there are **72, 057, 594, 037, 926, 936 combinations**.
- DES was cracked in **summer' 98** in **56 hours**.
- **This year** in January a message encrypted in DES was cracked in just **22 hours.**
- A **$20 million** investment in Application Specific Integrated Circuits can break a key every **six minutes**.

3.5.4 **DES enciphering algorithm:**



3.5.5 **Explanation of DES:** A detailed analysis of enciphering and deciphering process will be presented with a numerical example of a sixteen round DES system. The analysis will be done under the assumption that the 64-bit plain text

$X = (x_1, x_2, \ldots\ldots\ldots x_{64})$

= (0 1 2 3 4 5 6 7 8 9 A B C D E F)

in hexadecimal notation, enciphers into the 64-bit cipher text

$Y = (y_1, y_2, \ldots\ldots\ldots y_{64})$

Under the control of 64-bit externally entered key

$K = (k_1, k_2, \ldots\ldots\ldots k_{64})$

= (1 3 3 4 5 7 7 9 9 B B C D F F 1)

including 8 parity bits

3.5.5.1 **Key schedule: S**uppose the key is

K = (1 3 3 4 5 7 7 9 9 B B C D F F 1)

This 64-bit input key is expressed in binary notation as

K = (0001 0011 0011 0100 0101 0111 0111 1001
1001 1011 1011 1100 1101 1111 1111 0001)

The register contents Co (left) and Do (right) are determined by considering the key bits located at the position given in the table PC-1.

Table-1 Permuted Choice 1(PC-1)

57	49	41	33	25	17	9	1	58	50	42	34	26	18
10	2	59	51	43	35	27	19	11	3	60	52	44	36
63	55	47	39	31	23	15	7	62	54	46	38	30	22
14	6	61	53	45	37	29	21	13	5	28	20	12	4

Using the above table the blocks C_0 and D_0 are immediately obtained as shown below. The first two rows in the above table give the bit positions for the C_0 and the other two rows give the bit positions for D_0. The bit positions are obtained from the binary representation of the key.

C_0 = (1 1 1 1 0 0 0 0 1 1 0 0 1 1 0 0 1 0 1 0 1 0 1 0 1 1 1 1)

D_0 = (0 1 0 1 0 1 0 1 0 1 1 0 0 1 1 0 0 1 1 1 1 0 0 0 1 1 1 1)

Table-2 Shift Schedule for encipherment

Round#	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
No of Shifts	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Using Table-2, the blocks C_1 and D_1 are obtained from the blocks C_0 and D_0 , respectively, by shifting one place to the left as shown in the following.

C_1 = (1 1 1 0 0 0 0 1 1 0 0 1 1 0 0 1 0 1 0 1 0 1 0 1 1 1 1 1)
D_1 = (1 0 1 0 1 0 1 0 1 1 0 0 1 1 0 0 1 1 1 1 0 0 0 1 1 1 1 0)

Permuted choice 2 (PC 2) of Table-3 is the rule that defines how the 48 bit key vectors k_1, k_2, ……k_{16} are derived from the concatenated blocks (C_1, D_1), (C_2, D_2), ……(C_{16}, D_{16}) respectively.

Table-3 Permuted Choice 2 (PC-2)

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

The 48-bit key vector k1 is derived from (C_1, D_1) by taking the key bits located in Table-3.

K_1=(000110 110000 001011 101111 111111 000111
000001 110010)

Since the number of left shifts is 1 at round 2, the concatenated block (C_2, D_2) is created from the block (C_1, D_1) by shifting one place to the left as shown below.

C_2 = (1 1 0 0 0 0 1 1 0 0 1 1 0 0 1 0 1 0 1 0 1 0 1 1 1 1 1 1)
D_2 = (0 1 0 1 0 1 0 1 1 0 0 1 1 0 0 1 1 1 1 0 0 0 1 1 1 1 0 1)

Using PC-2, k_2 is easily obtained as

K_2 = (0111 1001 1010 1110 1101 1001 1101 1011 1100 1001
1110 0101)

Since the number of left shifts is 2 at round 3, the concatenated block (C_3, D_3) is created from the block (C_2, D_2) by shifting one place to the left as shown below.

C_3 = (0 0 0 0 1 1 0 0 1 1 0 0 1 0 1 0 1 0 1 0 1 1 1 1 1 1 1 1)
D_3 = (0 1 0 1 0 1 1 0 0 1 1 0 0 1 1 1 1 0 0 0 1 1 1 1 0 1 0 1)

Using PC-2, k_3 is easily obtained as

K_3 = (0101 0101 1111 1100 1000 1010 0100 0010 1100 1111
1001 1001)

Since the number of left shifts is 2 at round 4, the concatenated block (C_4, D_4) is created from the block (C_3, D_3) by shifting one place to the left as shown below.

C_4 = (0 0 1 1 0 0 1 1 0 0 1 0 1 0 1 0 1 0 1 1 1 1 1 1 1 1 0 0)
D_4 = (0 1 0 1 1 0 0 1 1 0 0 1 1 1 1 0 0 0 1 1 1 1 0 1 0 1 0 1)

Using PC-2, k_4 is easily obtained as

K_4 = (0111 0010 1010 1101 1101 0110 1101 1011 0011 0101
0001 1101)

Since the number of left shifts is 2 at round 5, the concatenated block (C_5, D_5) is created from the block (C_4, D_4) by shifting one place to the left as shown below.

C_5 = (1 1 0 0 1 1 0 0 1 0 1 0 1 0 1 0 1 1 1 1 1 1 1 1 0 0 0 0)
D_5 = (0 1 1 0 0 1 1 0 0 1 1 1 1 0 0 0 1 1 1 1 0 1 0 1 0 1 0 1)

Using PC-2, k_5 is easily obtained as

K_5=(0111 1100 1110 1100 0000 0111 1110 1011 0101 0011
1010 1000)

Since the number of left shifts is 2 at round 6, the concatenated block (C_6, D_6) is created from the block (C_5, D_5) by shifting one place to the left as shown below.

C_6 = (0 0 1 1 0 0 1 0 1 0 1 0 1 0 1 1 1 1 1 1 1 1 1 0 0 0 0 1 1)
D_6 = (1 0 0 1 1 0 0 1 1 1 1 0 0 0 1 1 1 1 0 1 0 1 0 1 0 1 0 1)

Using PC-2, k_6 is easily obtained as

K_6 = (0110 0011 1010 0101 0011 1110 0101 0000 0111 1011 0010 1111)

Since the number of left shifts is 2 at round 7, the concatenated block (C_7, D_7) is created from the block (C_6, D_6) by shifting one place to the left as shown below.

C_7 = (1 1 0 0 1 0 1 0 1 0 1 0 1 1 1 1 1 1 1 1 0 0 0 0 1 1 0 0)
D_7 = (0 1 1 0 0 1 1 1 1 0 0 0 1 1 1 1 0 1 0 1 0 1 0 1 0 1 1 0)

Using PC-2, k_7 is easily obtained as

K_7 = (1110 1100 1000 0100 1011 0111 1111 0110 0001 1000 1011 1100)

Since the number of left shifts is 2 at round 8, the concatenated block (C_8, D_8) is created from the block (C_7, D_7) by shifting one place to the left as shown below.

C_8 = (0 0 1 0 1 0 1 0 1 0 1 1 1 1 1 1 1 1 0 0 0 0 1 1 0 0 1 1)
D_8 = (1 0 0 1 1 1 1 0 0 0 1 1 1 1 0 1 0 1 0 1 0 1 0 1 1 0 0 1)

Using PC-2, k_8 is easily obtained as

K_8 = (1111 0111 1000 1010 0011 1010 1100 0001 0011 1011 1111 1011)

Since the number of left shifts is 1 at round 9, the concatenated block (C_9, D_9) is created from the block (C_8, D_8) by shifting one place to the left as shown below.

C_9 = (0 1 0 1 0 1 0 1 0 1 1 1 1 1 1 1 1 0 0 0 0 1 1 0 0 1 1 0)
D_9 = (0 0 1 1 1 1 0 0 0 1 1 1 1 0 1 0 1 0 1 0 1 0 1 1 0 0 1 1)

Using PC-2, k_9 is easily obtained as

K_9 = (1110 0000 1101 1011 1110 1011 1110 1101 1110 0111 1000 0001)

Since the number of left shifts is 2 at round 10, the concatenated block (C_{10}, D_{10}) is created from the block (C_9, D_9) by shifting one place to the left as shown below.

C_{10} = (0 1 0 1 0 1 0 1 1 1 1 1 1 1 1 0 0 0 0 1 1 0 0 1 1 0 0 1)

D_{10} = (1 1 1 1 0 0 0 1 1 1 1 0 1 0 1 0 1 0 1 0 1 1 0 0 1 1 0 0)

Using PC-2, k_{10} is easily obtained as

K_{10} = (1011 0001 1111 0011 0100 0111 1011 1010 0100 0110 0100 1111)

Since the number of left shifts is 2 at round 11, the concatenated block (C_{11}, D_{11}) is created from the block (C_{10}, D_{10}) by shifting one place to the left as shown below.

C_{11} = (0 1 0 1 0 1 1 1 1 1 1 1 0 0 0 0 1 1 0 0 1 1 0 0 1 0 1)
D_{11} = (1 1 0 0 0 1 1 1 1 0 1 0 1 0 1 0 1 0 1 1 0 0 1 1 0 0 1 1)

Using PC-2, k_{11} is easily obtained as

K_{11} = (0010 0001 0101 1111 1101 0011 1101 1110 1101 0011 1000 0110)

Since the number of left shifts is 2 at round 12, the concatenated block (C_{12}, D_{12}) is created from the block (C_{11}, D_{11}) by shifting one place to the left as shown below.

C_{12} = (0 1 0 1 1 1 1 1 1 1 1 0 0 0 0 1 1 0 0 1 1 0 0 1 0 1 0 1)
D_{12} = (0 0 0 1 1 1 1 0 1 0 1 0 1 0 1 0 1 1 0 0 1 1 0 0 1 1 1 1)

Using PC-2, k_{12} is easily obtained as

K_{12} = (0111 0101 0111 0001 1111 0101 1001 0100 0110 0111 1110 1001)

Since the number of left shifts is 2 at round 13, the concatenated block (C_{13}, D_{13}) is created from the block (C_{12}, D_{12}) by shifting one place to the left as shown below.

C_{13} = (0 1 1 1 1 1 1 1 1 0 0 0 0 1 1 0 0 1 1 0 0 1 0 1 0 1 0 1)
D_{13} = (0 1 1 1 1 0 1 0 1 0 1 0 1 0 1 1 0 0 1 1 0 0 1 1 1 1 0 0)

Using PC-2, k_{13} is easily obtained as

K_{13} = (1001 0111 1100 0101 1101 0001 1111 1010 1011 1010 0100 0001)

Since the number of left shifts is 2 at round 14, the concatenated block (C_{14}, D_{14}) is created from the block (C_{13}, D_{13}) by shifting one place to the left as shown below.

C_{14} = (1 1 1 1 1 1 1 0 0 0 0 1 1 0 0 1 1 0 0 1 0 1 0 1 0 1 0 1)
D_{14} = (1 1 1 0 1 0 1 0 1 0 1 0 1 1 0 0 1 1 0 0 1 1 1 1 0 0 0 1)

Using PC-2, k_{14} is easily obtained as

K_{14} = (0101 1111 0100 0011 1011 0111 1111 0010 1110 0111 0011 1010)

Since the number of left shifts is 2 at round 15, the concatenated block (C_{15}, D_{15}) is created from the block (C_{14}, D_{14}) by shifting one place to the left as shown below.

C_{15} = (1 1 1 1 1 0 0 0 0 1 1 0 0 1 1 0 0 1 0 1 0 1 0 1 0 1 1 1)
D_{15} = (1 0 1 0 1 0 1 0 1 0 1 1 0 0 1 1 0 0 1 1 1 1 0 0 0 1 1 1)

Using PC-2, k_{15} is easily obtained as

K_{15} = (1011 1111 1001 0001 1000 1101 0011 1101 0011 1111 0000 1010)

Since the number of left shifts is 1 at round 16, the concatenated block (C_{16}, D_{16}) is created from the block (C_{15}, D_{15}) by shifting one place to the left as shown below.

C_{16} = (1 1 1 1 0 0 0 0 1 1 0 0 1 1 0 0 1 0 1 0 1 0 1 0 1 1 1 1)
D_{16} = (0 1 0 1 0 1 0 1 0 1 1 0 0 1 1 0 0 1 1 1 1 0 0 0 1 1 1 1)

Using PC-2, k_{16} is easily obtained as

K_{16} = (1100 1011 0011 1101 1000 1011 0000 1110 0001 0111 1111 0101)

These internal keys k_1, k_2, ….k_{16} are used in round 1, 2 ….16 respectively for the purpose of encipherment. These 48-bits keys are generated through a series of permutations and left shifts of the 56 bits selected from the 64-bit external key.

3.5.5.2 **Encipherment:** The numerical computation is given first, along with a detailed explanation of how the algorithm is used for encipherment. The 64-bit plain text to be ciphered is

$X = (x_1, x_2, \ldots\ldots x_{64.})$
= (0 1 2 3 4 5 6 7 8 9 A B C D E F) in hexadecimal notation.
= (0000 0001 0010 0011 0100 0101 0110 0111
1000 1001 1010 1011 1100 1101 1110 1111)

This plain text X is first subjected to an initial permutation (IP) to make it split into two blocks L_0 (left) and R_0 (right) where each of them consists of X/2 = 32 bits as indicated in Table-4. where (L_0,R_0) denotes a concatenation consisting of the bits of L_0 followed by R_0.

Table-4 Initial Permutation (IP)

L_0	58	50	42	34	26	18	10	2
	60	52	44	36	28	20	12	4
	62	54	46	38	30	22	14	6
	64	56	48	40	32	24	16	8
R_0	57	49	41	33	25	17	9	1
	59	51	43	35	27	19	11	3
	61	53	45	37	29	21	13	5
	63	55	47	39	31	23	15	7

L_0 = (1100 1100 0000 0000 1100 1100 1111 1111)
R_0 = (1111 0000 1010 1010 1111 0000 1010 1010)

This right half of the plain text to round 0, R_0 is expanded from 32 bits to 48 bits according to Table-5.

Table-5 E Bit-Selection Table

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

The symbol E of $E(R_0)$ denotes a function, which takes a block of 32 bits as its input and yields one of 48 bits as its output. Table-5 consists of eight blocks of 6 bits each, but the central portion represents the data and the first and last columns denote appendices for expansion. Hence, we have

$E(R_0)$ = (011110 100001 010101 010101 011110 100001
010101 010101)

Thus, the 32-bit R_0 can be spread out and scrambled into 48 bits with the E-Table.

The key dependent function, $T_j = E(R_0) \oplus k_j$, $0 < I < 15$,, $1 < j < 16$ can be computed in terms of the E bit selection $E(R_0)$, $0 < I < 15$ and the key schedule k_j, $1 < j < 16$. the cipher function f_j, $1 < I < 16$, is defined in terms of T_j and the permutation function P(Bj)

Once $E(R_0)$ is generated, it is added bit by bit to k1 and 48-bit vector is resulted in

$T_1 = E(R_0) \oplus k_1$
= (011110 100001 010101 010101 011110 100001 010101 010101)
$\oplus$(000110 110000 001011 101111 111111 000111 000001 110010)
= (011000 010001 011110 111010 100001 100110 010100 100111)

Where $\oplus$ denotes bit-by-bit addition modulo-2. this 48-bit input T_1 to the S-boxes is passed through a non-linear S-box transformation to form the 32-bit output.

Table-6 Primitive S –Box Functions

S1															
14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

S2															
15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	13

S3															
10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

S4															
7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

S5															
2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

S6															
12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

S7															
4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

S8															
13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

The elements of vector T_j , are used as arguments in the substitution operations (S-boxes) S1 through S8. Each S-box is described as a matrix of four rows and 16 columns as shown in Table-6. Grouping the 48-bit T_1 into sets of 6 bits , as expressed above, leads to easy computation of the substitution operations for S1 through S8 as follows.

T_1 = (011000 010001 011110 111010 100001 100110 010100 100111)

$S^{00}{}_1(1100) = S^0{}_1(12) = 5 = 0101$
$S^{01}{}_2(1000) = S^1{}_2(8) = 12 = 1100$
$S^{00}{}_3(1111) = S^0{}_3(15) = 8 = 1000$
$S^{10}{}_4(1101) = S^2{}_4(13) = 2 = 0010$
$S^{11}{}_5(0000) = S^3{}_5(0) = 11 = 1011$
$S^{10}{}_6(0011) = S^2{}_6(3) = 5 = 0101$
$S^{00}{}_7(1010) = S^{00}{}_7(10) = 9 = 1001$
$S^{11}{}_8(0011) = S^{11}{}_8(3) = 7 = 0111$

Each member in the above equation represents the 4-bit output of an individual S-box, and concatenating all of these outputs yields the 32 bits represented by vector B_1.

B_1 = (0101 1100 1000 0010 1011 0101 1001 0111)

The permutation function $P(B_1)$ yields a 32-bit output from a 32-bit input by permuting the bits of the above equation. Such a function is defined in Table-7.

Table-7 Permutation Function p

16	7	20	21
29	12	28	17
1	15	23	26
5	18	31	10
2	8	24	14
32	27	3	9
19	13	30	6
22	11	4	25

$P(B_1)$ = (0010 0011 0100 1010 1010 1001 1011 1011)

Modulo 2 addition of $P(B_1)$ with L_0 may be expressed as

$R_1 = P(B_1) \oplus L_0$

= (0010 0011 0100 1010 1010 1001 1011 1011)

$\oplus$(1100 1100 0000 00001100 1100 1111 1111)

= (1110 1111 0100 1010 0110 0101 0100 0100)

This is the result representing the right half output after round one. Since $L_1 = R0$, the left half output after round one is immediately obtained from the equation

L_1 = (1111 0000 1010 1010 1111 0000 1010 1010)

Thus, the computation of R_1 and L_1 completes the first round encipherment.

Let us now consider the second round encipherment. Expanding R_1 with the help of table-5,we have

$E(R_1)$ = (011101 011110 101001 010100 001100 001010
101000 001001)

Modulo-2 addition of $E(R_1)$ with k_2 yields

$T_1 = E(R_1) \oplus k_2$

= (011101 011110 101001 010100 001100 001010 101000 001001)

$\oplus$(011110 011010 111011 011001 110110 111100 100111 100101)

= (000011 000100 010010 001101 111010 110110 001111 101100)

The substitution operations are followed by

$S^{01}{}_1(0001) = S^0{}_1(1) = 15 = 1111$

$S^{00}{}_2(0010) = S^1{}_2(2\ ) = 8 = 1000$

$S^{00}{}_3(1001) = S^0{}_3(9) = 13 = 1101$

$S^{01}{}_4(0110) = S^2{}_4(6) = 0 = 0000$

S^{10}_5 (1101) = S^3_5 (13) = 3 = 0011
S^{10}_6 (1011) = S^2_6 (11) = 10 = 1010
S^{01}_7 (0111) = S^{00}_7 (7) = 10 = 1010
S^{10}_8 (0110) = S^{11}_8 (6) = 14 = 1110

Concatenating all these results we get

B_2 = (1111 1000 1101 0000 0011 1010 1010 1110)

Using Table-7, the permutation $P(B_2)$ becomes

$P(B_2)$ = (00111100101010111000011110100011)

Thus, the right-half output after round two is obtained by modulo2 addition of $P(B_2)$with L_1 such that

$R_2 = P(B_2) \oplus L_1$
= (0011 1100 1010 1011 1000 0111 1010 0011)
$\oplus$ (1111 0000 1010 1010 1111 0000 1010 1010)
R_2 = (1100 1100 0000 0001 0111 0111 0000 1001)

The left-half output after round two becomes

$L_2 = R_1$
= (1110 1111 0100 1010 0110 0101 0100 0100)

Let us now consider the third round encipherment. Expanding R_2 with the aid of Table-5, we have

R_2 = 1100 1100 0000 0001 0111 0111 0000 1001

$E(R_2)$ = (111001 011000 000000 000010 101110 101110 100001 010011)

Modulo-2 addition of $E(R_2)$ with K_3 yields

T_2 == $E(R_2) \oplus K_3$
= (111001 011000 000000 000010 101110 101110 100001 010011)
$\oplus$ (010101 011111 110010 001010 010000 101100 111110 011001)
= (101100 000111 110010 001000 111110 000010 011111 001010)

The substitution operations for S-boxes are fol1owed by

S^{10}_1 (0110) = S^2_1 (6) = 2 = 0010
S^{01}_2 (0011) = S^1_2 (3) = 7 = 0111
S^{10}_3 (1001) = S^2_3 (9) = 1 = 0001
S^{00}_4 (0100) = S^0_4 (4) = 0 = 0000
S^{10}_5 (1111) = S^2_5 (15) = 14 = 1110
S^{00}_6 (0001) = S^0_6 (1) = 1 = 0001

$S^{01}_7(1111) = S^1_7(15) = 6 = 0110$

$S^{00}_8(0101) = S^0_8(5) = 15 = 1111$

Concatenating all these results we get

$B_3 = (0010\ 0111\ 0001\ 0000\ 1110\ 0001\ 0110\ 1111)$

Using Table-7, the permutation $P(B_3)$ becomes

$P(B_3) = (0100\ 1101\ 0001"0110\ 0110\ 1110\ 1011\ 0000)$

Thus, the right-half output after round three is obtained by modulo-2 addition of $P(B_3)$ with L_2 such that

$R_3 = P(B_3) \oplus L_2$

$= (0100\ 1101\ 0001\ 0110\ 0110\ 1110\ 1011\ 0000)$

$\oplus (1110\ 1111\ 0100\ 1010\ 0110\ 0101\ 0100\ 0100)$

$R_3 = (1010\ 0010\ 0101\ 1100\ 0000\ 1011\ 1111\ 0100)$

The left-half output after round three becomes

$L_3 = R_2$

$=(1100\ 1100\ 0000\ 0001\ 0111\ 0111\ 0000\ 1001)$

Concatenation of R_3 with L_3 is called the pre output block in our sixteen round cipher system.

Let us now consider the fourth round encipherment. Expanding R_3 with the aid of Table-5, we have

$R_3 = (1010\ 0010\ 0101\ 1100\ 0000\ 1011\ 1111\ 0100)$

$E(R_3) = (010100\ 000100\ 001011\ 111000\ 000001\ 010111\ 111110\ 101001)$

Modulo-2 addition of $E(R_3)$ with K_4 yields

$T_3 = E(R_3) \oplus K_4$

$= (010100\ 000100\ 001011\ 111000\ 000001\ 010111\ 111110\ 101001)$

$\oplus (011100\ 101010\ 110111\ 010110\ 110110\ 110011\ 010100\ 011101)$

$= (001000\ 101110\ 111100\ 101110\ 110111\ 100100\ 101010\ 110100)$

The substitution operations for S-boxes are followed by

$S^{00}_1 \quad (0100) = S^0_1 \quad (4) \quad = 2 \quad = 0010$

$S^{10}_2 \quad (0111) = S^2_2 \quad (7) \quad = 1 \quad = 0001$

$S^{10}_3 \quad (1110) = S^2_3 \quad (14) \quad = 14 \quad = 1110$

$S^{10}_4 \quad (0111) = S^2_4 \quad (7) \quad = 13 \quad = 1101$

S^{11}_5 (1011) $= S^3_5$ (11) = 9 = 1001
S^{10}_6 (0010) $= S^2_6$ (2) = 15 = 1111
S^{10}_7 (0101) $= S^2_7$ (5) = 3 = 0011
S^{10}_8 (1010) $= S^2_8$ (10) = 10 = 1010

Concatenating all these results we get

B_4 = (0010 0001 1110 1101 1001 1111 0011 1010)

Using Table-7, the permutation $P(B_4)$ becomes

$P(B_4)$ = (1011 loll 0010 0011 0111 0111 0100 1100)

Thus, the right-half output after round four is obtained by modulo-2 addition of $P(B_4)$ with L_3 such that

$R_4 = P(B_4) \oplus L_3$
= (1011 1011 0010 0011 0111 0111 0100 1100)
$\oplus$ (1100 1100 0000 0001 0111 0111 0000 1001)
= (0111 0111 0010 0010 0000 0000 0100 0101)

The left-half output after round three becomes

$L_4 = R_3$
= (1010 0010 0101 1100 0000 1011 1111 0100)

Concatenation of R_4 with L_4 is called the pre output block in our sixteen round cipher system.

Let us now consider the fifth round encipherment. Expanding R_4 with the aid of Table-5, we have

R_4 = (0111 0111 0010 0010 0000 0000 0100 0101)
$E(R_4)$ = (101110 101110 100100 000100 000000 000000 001000 001010)

Modulo-2 addition of $E(R_4)$ with K_5 yields

$T_4 = E(R_4) \oplus K_5$
=(101110 101110 100100 000100 000000 000000 001000 001010)
$\oplus$(011111 001110 110000 000111 111010 110101 001110 101000)
=(110001 100000 010100 000011 111010 110101 000110 100010)

The substitution operations for S-boxes are followed by

$S^{11}_1(1000) = S^3_1$ (8) = 5 = 0101
$S^{10}_2(0000) = S^2_2$ (0) = 0 = 0000
$S^{00}_3(1010) = S^0_3$ (10) = 12 = 1100
$S^{01}_4(0001) = S^1_4$ (1) = 8 = 1000
$S^{10}_5(1101) = S^2_5$ (13) = 3 = 0011

$S^{11}{}_6(1010) = S^3{}_6\ (10) = 1 = 0001$
$S^{00}{}_7(0011) = S^0{}_7\ (3) = 14 = 1110$
$S^{10}{}_8(0001) = S^2{}_8\ (1) = 11 = 1011$

Concatenating all these results we get

B_5 = (0101 0000 1100 1000 0011 0001 l110 1011)

Using Table-7, the permutation $P(B_5)$ becomes

$P(B_5)$ = (0010 1000 0001 0011 1010 1101 1100 0011)

Thus, the right-half output after round five is obtained by modulo-2 addition of $P(B_5)$ with L_4 such that

$R_5 = P(B_5) \oplus L_4$
= (0010 1000 0001 0011 1010 1101 1100 0011)
$\oplus$ (1010 0010 0101 1100 0000 1011 1111 0100)
R_5 = (1000 1010 0100 1111 1010 0110 0011 0111)

The left-half output after round three becomes

$L_5 = R_4$
= (0111 0111 0010 0010 0000 0000 0100 0101)

Concatenation of R_5 with L_5 is called the pre output block in our sixteen round cipher system.

Let us now consider the sixth round encipherment. Expanding R_5 with the aid of Table-5, we have

R_5=(1000 1010 0100 1111 1010 0110 0011 0111)
$E(R_5)$ = (110001 010100 001001 011111 110100 001100 000110 101111)

Modulo-2 addition of $E(R_4)$ with K_5 yields

$T_5 = E(R_5) \oplus K_6$
=(110001 010100 001001 011111 110100 001100 000110 101111)
$\oplus$(011000 111010 010100 111110 010100 000111 101100 101111)
=(101001 101110 011101 100001 100000 001011 101010 000000)

The substitution operations for S-boxes are followed by

$S^{11}{}_1\ (0100) = S^3{}_1\ (4) = 4 = 0100$
$S^{10}{}_2\ (0111) = S^2{}_2\ (7) = 1 = 0001$

S^{01}_3 (1110)=S^1_3(14) = 15= 1111
S^{11}_4 (0000)= S^3_4(0) =3 = 0011
S^{10}_5 (0000)=S^2_5(0) =4 = 0100
S^{01}_6 (0101)=S^1_6(5) = 12 = 1100
S^{10}_7 (0101)=S^2_7(5) =3 = 0011
S^{00}_8 (0000)= S^0_8(0) = 13 = 1101

Concatenating all these results we get

B_6 = (1111 1000 1101 0000 0011 1010 1010 1110)

Using Table-7, the permutation $P(B_6)$ becomes

$P(B_6)$ = (00111100101010111000011110100011)

Thus, the right-half output after round two is obtained by modulo2 addition of $P(B_6)$with L_5 such that

$R_6 = P(B_6) \oplus L_5$

= (0011 1100 1010 1011 1000 0111 1010 0011)

$\oplus$ (1111 0000 1010 1010 1111 0000 1010 1010)

R_6 = (1100 1100 0000 0001 0111 0111 0000 1001)

The left-half output after round two becomes

$L_6 = R_5$

= (1110 1111 0100 1010 0110 0101 0100 0100)

Let us now consider the third round encipherment. Expanding R_6 with the aid of Table-5, we have

R_6= 1100 1100 0000 0001 0111 0111 0000 1001

$E(R_6)$ = (111001 011000 000000 000010 101110 101110 100001 010011)

Modulo-2 addition of $E(R_6)$ with k_7 yields

$T_6 = E(R_6) \oplus K_7$

= (111001 011000 000000 000010 101110 101110 100001 010011)

$\oplus$ (010101 011111 110010 001010 010000 101100 111110 011001)

= (101100 000111 110010 001000 111110 000010 011111 001010)

The substitution operations for S-boxes are fol1owed by

S^{10}_1 (0110) = S^2_1 (6) = 2 = 0010

S^{01}_2 (0011) = S^1_2 (3) = 7 = 0111

S^{10}_3 (1001) = S^2_3 (9) = 1 = 0001

$S^{00}{}_4\,(0100) = S^0{}_4\,(4) = 0 = \quad 0000$
$S^{10}{}_5\,(1111) = S^2{}_5\,(15) = 14 = 1110$
$S^{00}{}_6\,(0001) = S^0{}_6\,(1) = 1 = 0001$
$S^{01}{}_7\,(1111) = S^1{}_7\,(15) = 6 = 0110$
$S^{00}{}_8\,(0101) = S^0{}_8\,(5) = 15 = 1111$

Concatenating all these results we get

B_7 = (0001 0000 0111 0101 0100 0000 1010 1101)

Using Table-7, the permutation $P(B_7)$ becomes

$P(B_7)$ = (1000 1100 0000 0101 0001 1100 0010 0111)

Thus, the right-half output after round seven is obtained by modulo-2 addition of $P(B_7)$ with L_6 such that

$R_7 = P(B_7) \oplus L_6$

= (1000 1100 0000 0101 0001 1100 0010 0111)

$\oplus$ (1000 1010 0100 1111 1010 0110 0011 0111)

R_7 = (0000 0110 0100 1010 1011 1010 0001 0000)

The left-half output after round three becomes

$L_7 = R_6$

= (1110 1001 0110 0111 1100 1101 0110 1001)

Concatenation of R_7 with L_7 is called the pre output block in our sixteen round cipher system.

Let us now consider the eighth round encipherment. Expanding R_7 with the aid of Table-5, we have

R_7 = (0000 0110 0100 1010 1011 1010 0001 0000)

$E(R_7)$=(000000 001100 001001 010101 010111 110100 000010 100000)

Modulo-2 addition of $E(R_7)$ with K_8 yields

$T_7 = E(R_7) \oplus K_8$

= (000000 001100 001001 010101 010111 110100 000010 100000)

$\oplus$ (111101 111000 101000 111010 110000 010011 101111 111011)

= (111101 110100 100001 101111 100111 100111 101101 011011)

The substitution operations for S-boxes are followed by

$S^{11}{}_1(1110) = S^3{}_1(14) \quad = 6 \quad = 0110$

$S^{10}{}_2(1010) = S^2{}_2(10) \quad = 12 \quad = 1100$

$S^{11}{}_3(0000) = S^3{}_3(0) = 1 = 0001$
$S^{11}{}_4(0111) = S^3{}_4(7) = 8 = 1000$
$S^{11}{}_5(0011) = S^3{}_5(3) = 7 = 0111$
$S^{11}{}_6(0011) = S^3{}_6(3) = 12 = 1100$
$S^{11}{}_7(0110) = S^3{}_7(6) = 10 = 1010$
$S^{01}{}_8(1101) = S^1{}_8(13) = 14 = 1110$

Concatenating all these results we get

B_8 = (0101 0000 1100 1000 0011 0001 l110 1011)

Using Table-7, the permutation $P(B_8)$ becomes

$P(B_8)$ = (0010 1000 0001 0011 1010 1101 1100 0011)

Thus, the right-half output after round five is obtained by modulo-2 addition of $P(B_8)$ with L_7 such that

$R_7 = P(B_8) \oplus L_7$
= (0010 1000 0001 0011 1010 1101 1100 0011)
$\oplus$ (1010 0010 0101 1100 0000 1011 1111 0100)
R_7 = (1000 1010 0100 1111 1010 0110 0011 0111)

The left-half output after round three becomes

$L_8 = R_7$
= (0111 0111 0010 0010 0000 0000 0100 0101)

Concatenation of R_8 with L_8 is called the pre output block in our sixteen round cipher system.

Let us now consider the sixth round encipherment. Expanding R_8 with the aid of Table-5, we have

R_8=(1000 1010 0100 1111 1010 0110 0011 0111)

$E(R_8)$ = (110001 010100 001001 011111 110100 001100 000110 101111)

Modulo-2 addition of $E(R_8)$ with K_9 yields

$T_8 = E(R_8) \oplus K_9$
=(110001 010100 001001 011111 110100 001100 000110 101111)
$\oplus$(011000 111010 010100 111110 010100 000111 101100 101111)
=(101001 101110 011101 100001 100000 001011 101010 000000)

The substitution operations for S-boxes are followed by

S^{11}_1 (0100)= S^3_1 (4) = 4 = 0100
S^{10}_2 (0111)= S^2_2 (7) =1 = 0001
S^{01}_3 (1110)=S^1_3(14) = 15 = 1111
S^{11}_4 (0000)= S^3_4(0) =3 = 0011
S^{10}_5 (0000)=S^2_5(0) =4 = 0100
S^{01}_6 (0101)=S^1_6(5) = 12 = 1100
S^{10}_7 (0101)=S^2_7(5) =3 = 0011
S^{00}_8 (0000)= S^0_8(0) = 13 = 1101

Concatenating all these results we get

B_9 = (0010 0111 0001 0000 1110 0001 0110 1111)

Using Table-7, the permutation $P(B_9)$ becomes

$P(B_9)$ = (0100 1101 0001"0110 0110 1110 1011 0000)

Thus, the right-half output after round three is obtained by modulo-2 addition of $P(B_9)$ with L_8 such that

$R_9 = P(B_9) \oplus L_8$
= (0100 1101 0001 0110 0110 1110 1011 0000)
$\oplus$ (1110 1111 0100 1010 0110 0101 0100 0100)
R_9 = (1010 0010 0101 1100 0000 1011 1111 0100)

The left-half output after round three becomes

$L_9 = R_8$
=(1100 1100 0000 0001 0111 0111 0000 1001)

Concatenation of R_9 with L_9 is called the pre output block in our sixteen round cipher system.

Let us now consider the fourth round encipherment. Expanding R_9 with the aid of Table-5, we have

R_9 = (1010 0010 0101 1100 0000 1011 1111 0100)
$E(R_9)$ = (010100 000100 001011 111000 000001 010111 111110 101001)

Modulo-2 addition of $E(R_9)$ with K_{10} yields

$T_9 = E(R_9) \oplus K_{10}$
= (010100 000100 001011 111000 000001 010111 111110 101001)
$\oplus$ (011100 101010 110111 010110 110110 110011 010100 011101)
= (001000 101110 111100 101110 110111 100100 101010 110100)

The substitution operations for S-boxes are followed by

S^{00}_1 (0100) $= S^0_1$ (4) = 2 = 0010
S^{10}_2 (0111) $= S^2_2$ (7) = 1 = 0001
S^{10}_3 (1110) $= S^2_3$ (14) = 14 = 1110
S^{10}_4 (0111) $= S^2_4$ (7) = 13 = 1101
S^{11}_5 (1011) $= S^3_5$ (11) = 9 = 1001
S^{10}_6 (0010) $= S^2_6$ (2) = 15 = 1111
S^{10}_7 (0101) $= S^2_7$ (5) = 3 = 0011
S^{10}_8 (1010) $= S^2_8$ (10) = 10 = 1010

Concatenating all these results we get

B_{10} = (0010 0001 1110 1101 1001 1111 0011 1010)

Using Table-7, the permutation $P(B_{10})$ becomes

$P(B_{10})$ = (1011 loll 0010 0011 0111 0111 0100 1100)

Thus, the right-half output after round four is obtained by modulo-2 addition of $P(B_{10})$ with L_9 such that

$R_{10} = P(B_{10}) \oplus L_9$
= (1011 1011 0010 0011 0111 0111 0100 1100)
$\oplus$ (1100 1100 0000 0001 0111 0111 0000 1001)
= (0111 0111 0010 0010 0000 0000 0100 0101)

The left-half output after round three becomes

$L_{10} = R_9$
= (1010 0010 0101 1100 0000 1011 1111 0100)

Concatenation of R_{10} with L_{10} is called the pre output block in our sixteen round cipher system.

Let us now consider the fifth round encipherment. Expanding R_{10} with the aid of Table-5, we have

R_{10} = (0111 0111 0010 0010 0000 0000 0100 0101)
$E(R_{10})$ = (101110 101110 100100 000100 000000 000000 001000 001010)

Modulo-2 addition of $E(R_{10})$ with K_{11} yields

$T_{10} = E(R_{10}) \oplus K_{11}$
=(101110 101110 100100 000100 000000 000000 001000 001010)
$\oplus$(011111 001110 110000 000111 111010 110101 001110 101000)
=(110001 100000 010100 000011 111010 110101 000110 100010)

The substitution operations for S-boxes are followed by

$S^{11}{}_1(1000) = S^3{}_1$ (8) = 5 = 0101
$S^{10}{}_2(0000) = S^2{}_2$ (0) = 0 = 0000
$S^{00}{}_3(1010) = S^0{}_3$ (10) = 12 = 1100
$S^{01}{}_4(0001) = S^1{}_4$ (1) = 8 = 1000
$S^{10}{}_5(1101) = S^2{}_5$ (13) = 3 = 0011
$S^{11}{}_6(1010) = S^3{}_6$ (10) = 1 = 0001
$S^{00}{}_7(0011) = S^0{}_7$ (3) = 14 = 1110
$S^{10}{}_8(0001) = S^2{}_8$ (1) = 11 = 1011

Concatenating all these results we get

B_{11} = (0010 0111 0001 0000 1110 0001 0110 1111)

Using Table-7, the permutation $P(B_{11})$ becomes

$P(B_{11})$ = (0100 1101 0001"0110 0110 1110 1011 0000)

Thus, the right-half output after round three is obtained by modulo-2 addition of $P(B_{11})$ with L_{10} such that

$R_{11} = P(B_{11}) \oplus L_{10}$
= (0100 1101 0001 0110 0110 1110 1011 0000)
$\oplus$ (1110 1111 0100 1010 0110 0101 0100 0100)
R_{11} = (1010 0010 0101 1100 0000 1011 1111 0100)

The left-half output after round three becomes

$L_{11} = R_{10}$
= (1100 1100 0000 0001 0111 0111 0000 1001)

Concatenation of R_{11} with L_{11} is called the pre output block in our sixteen round cipher system.

Let us now consider the fourth round encipherment. Expanding R_{11} with the aid of Table-5, we have

R_{11} = (1010 0010 0101 1100 0000 1011 1111 0100)
$E(R_{11})$ = (010100 000100 001011 111000 000001 010111 111110 101001)

Modulo-2 addition of $E(R_{11})$ with K_{12} yields

$T_{11} = E(R_{11}) \oplus K_{12}$
= (010100 000100 001011 111000 000001 010111 111110 101001)
$\oplus$ (011100 101010 110111 010110 110110 110011 010100 011101)
= (001000 101110 111100 101110 110111 100100 101010 110100)

The substitution operations for S-boxes are followed by

S^{00}_1 (0100) = S^0_1 (4) = 2 = 0010
S^{10}_2 (0111) = S^2_2 (7) = 1 = 0001
S^{10}_3 (1110) = S^2_3 (14) = 14 = 1110
S^{10}_4 (0111) = S^2_4 (7) = 13 = 1101
S^{11}_5 (1011) = S^3_5 (11) = 9 = 1001
S^{10}_6 (0010) = S^2_6 (2) = 15 = 1111
S^{10}_7 (0101) = S^2_7 (5) = 3 = 0011
S^{10}_8 (1010) = S^2_8 (10) = 10 = 1010

Concatenating all these results we get

B_{12} = (1111 1000 1101 0000 0011 1010 1010 1110)

Using Table-7, the permutation $P(B_{12})$ becomes

$P(B_{12})$ = (00111100101010111000011110100011)

Thus, the right-half output after round two is obtained by modulo2 addition of $P(B_{12})$with L_{11} such that

$R_{12} = P(B_{12}) \oplus L_{11}$
= (0011 1100 1010 1011 1000 0111 1010 0011)
$\oplus$ (1111 0000 1010 1010 1111 0000 1010 1010)
R_{12} = (1100 1100 0000 0001 0111 0111 0000 1001)

The left-half output after round two becomes

$L_{12} = R_{11}$
= (1110 1111 0100 1010 0110 0101 0100 0100)

Let us now consider the third round encipherment. Expanding R_{12} with the aid of Table-5, we have

R_{12}= 1100 1100 0000 0001 0111 0111 0000 1001
$E(R_{12})$ = (111001 011000 000000 000010 101110 101110 100001 010011)

Modulo-2 addition of $E(R_{12})$ with k_{13} yields

$T_{12} = E(R_{12}) \oplus K_{13}$

(111001 011000 000000 000010 101110 101110 100001 010011)

$\oplus$(010101 011111 110010 001010 010000 101100 111110 011001)

= (101100 000111 110010 001000 111110 000010 011111 001010)

The substitution operations for S-boxes are fol1owed by

S^{10}_1 (0110) = S^2_1 (6) = 2 = 0010

S^{01}_2 (0011) = S^1_2 (3) = 7 = 0111

S^{10}_3 (1001) = S^2_3 (9) = 1 = 0001

S^{00}_4 (0100) = S^0_4 (4) = 0 = 0000

S^{10}_5 (1111) = S^2_5 (15) = 14 = 1110

S^{00}_6 (0001) = S^0_6 (1) = 1 = 0001

S^{01}_7 (1111) = S^1_7 (15) = 6 = 0110

S^{00}_8 (0101) = S^0_8 (5) = 15 = 1111

Concatenating all these results we get

B_{13} = (0101 0000 1100 1000 0011 0001 l110 1011)

Using Table-7, the permutation $P(B_{13})$ becomes

$P(B_{13})$ = (0010 1000 0001 0011 1010 1101 1100 0011)

Thus, the right-half output after round five is obtained by modulo-2 addition of $P(B_{13})$ with L_{12} such that

$R_{13} = P(B_{13}) \oplus L_{12}$

= (0010 1000 0001 0011 1010 1101 1100 0011)

$\oplus$ (1010 0010 0101 1100 0000 1011 1111 0100)

R_{13} =(1000 1010 0100 1111 1010 0110 0011 0111)

The left-half output after round three becomes

$L_{13} = R_7$

= (0111 0111 0010 0010 0000 0000 0100 0101)

Concatenation of R_{13} with L_{13} is called the pre output block in our sixteen round cipher system.

Let us now consider the sixth round encipherment. Expanding R_{13} with the aid of Table-5, we have

R_{13}=(1000 1010 0100 1111 1010 0110 0011 0111)

$E(R_{13})$ = (110001 010100 001001 011111 110100 001100 000110 101111)

Modulo-2 addition of $E(R_{13})$ with K_{14} yields

$T_{13} = E(R_{13}) \oplus K_{14}$

=(110001 010100 001001 011111 110100 001100 000110 101111)

$\oplus$(011000 111010 010100 111110 010100 000111 101100 101111)

=(101001 101110 011101 100001 100000 001011 101010 000000)

The substitution operations for S-boxes are followed by

S^{11}_{1} (0100)= S^{3}_{1} (4) = 4 = 0100

S^{10}_{2} (0111)= S^{2}_{2} (7) =1 = 0001

S^{01}_{3} (1110)=S^{1}_{3}(14) = 15 = 1111

S^{11}_{4} (0000)= S^{3}_{4}(0) =3 = 0011

S^{10}_{5} (0000)=S^{2}_{5}(0) =4 = 0100

S^{01}_{6} (0101)=S^{1}_{6}(5) = 12 = 1100

S^{10}_{7} (0101)=S^{2}_{7}(5) =3 = 0011

S^{00}_{8} (0000)= S^{0}_{8}(0) = 13 = 1101

Concatenating all these results we get

B_{14} = (1111 1000 1101 0000 0011 1010 1010 1110)

Using Table-7, the permutation $P(B_{14})$ becomes

$P(B_{14})$ = (00111100101010111000011110100011)

Thus, the right-half output after round two is obtained by modulo2 addition of $P(B_{14})$with L_{13} such that

$R_{14} = P(B_{14}) \oplus L_{13}$

= (0011 1100 1010 1011 1000 0111 1010 0011)

$\oplus$ (1111 0000 1010 1010 1111 0000 1010 1010)

R_{14} = (1100 1100 0000 0001 0111 0111 0000 1001)

The left-half output after round two becomes

$L_{14} = R_{13}$

= (1110 1111 0100 1010 0110 0101 0100 0100)

Let us now consider the third round encipherment. Expanding R_{14} with the aid of Table-5, we have

R_{14}= 1100 1100 0000 0001 0111 0111 0000 1001

$E(R_{14})$ = (111001 011000 000000 000010 101110 101110 100001 010011)

Modulo-2 addition of $E(R_{14})$ with k_{15} yields

T_{14}= $E(R_{14}) \oplus K_{15}$

= (111001 011000 000000 000010 101110 101110 100001 010011)

$\oplus$ (010101 011111 110010 001010 010000 101100 111110 011001)

= (101100 000111 110010 001000 111110 000010 011111 001010)

The substitution operations for S-boxes are fol1owed by

$S^{10}_1(0110) = S^2_1(6) = 2 =$ 0010

$S^{01}_2(0011) = S^1_2(3) = 7 =$ 0111

$S^{10}_3(1001) = S^2_3(9) = 1 =$ 0001

$S^{00}_4(0100) = S^0_4(4) = 0 =$ 0000

$S^{10}_5(1111) = S^2_5(15) = 14 =$ 1110

$S^{00}_6(0001) = S^0_6(1) = 1 =$ 0001

$S^{01}_7(1111) = S^1_7(15) = 6 =$ 0110

$S^{00}_8(0101) = S^0_8(5) = 15 =$ 1111

Concatenating all these results we get

B_{15} = (1111 1000 1101 0000 0011 1010 1010 1110)

Using Table-7, the permutation $P(B_{15})$ becomes

$P(B_{15})$ = (00111100101010111000011110100011)

Thus, the right-half output after round two is obtained by modulo2 addition of $P(B_{15})$with L_{14} such that

$R_{15} = P(B_{15}) \oplus L_{14}$

= (0011 1100 1010 1011 1000 0111 1010 0011)

$\oplus$ (1111 0000 1010 1010 1111 0000 1010 1010)

R_{15} = (1100 1100 0000 0001 0111 0111 0000 1001)

The left-half output after round two becomes

$L_{15} = R_5$

= (1110 1111 0100 1010 0110 0101 0100 0100)

Let us now consider the third round encipherment. Expanding R_{15} with the aid of Table-5, we have

R_{15}= 1100 1100 0000 0001 0111 0111 0000 1001

$E(R_{15})$ = (111001 011000 000000 000010 101110 101110 100001 010011)

Modulo-2 addition of $E(R_{15})$ with k_{16} yields

T_{15}= $E(R_{15}) \oplus K_{16}$

= (111001 011000 000000 000010 101110 101110 100001 010011)

⊕(010101 011111 110010 001010 010000 101100 111110 011001)

= (101100 000111 110010 001000 111110 000010 011111 001010)

The substitution operations for S-boxes are fol1owed by

$S^{10}_1(0110) = S^2_1(6) = 2 = 0010$
$S^{01}_2(0011) = S^1_2(3) = 7 = 0111$
$S^{10}_3(1001) = S^2_3(9) = 1 = 0001$
$S^{00}_4(0100) = S^0_4(4) = 0 = 0000$
$S^{10}_5(1111) = S^2_5(15) = 14 = 1110$
$S^{00}_6(0001) = S^0_6(1) = 1 = 0001$
$S^{01}_7(1111) = S^1_7(15) = 6 = 0110$
$S^{00}_8(0101) = S^0_8(5) = 15 = 1111$

Concatenating all these results we get

B_{16} = (1010 0111 1000 0011 0010 0100 0010 1001)

Using Table-7, the permutation $P(B_{16})$ becomes

$P(B_{16})$ = (1100 1000 1100 0000 0100 1111 1001 1000)

Thus, the right-half output after round 16 is obtained by modulo-2 addition of $P(B_{16})$ with L_{15} such that

$R_{16} = P(B_{16}) \oplus L_{15}$

= (1100 1000 1100 0000 0100 1111 1001 1000)

⊕ (1100 0010 1000 1100 1001 0110 0000 1101)

R_{16} = (0000 1010 0100 1100 1101 1001 1001 0101)

The left-half output after round 16 becomes

$L_{16} = R_{15}$

= (0100 0011 0100 0010 0011 0010 0011 0100)

$IP^{\wedge(-1)}$ = (1000 0101 1110 1000 0001 0011 0101 0100
0000 1111 0000 1010 1011 0100 0000 0101)

Concatenation of R_{16} with L_{16} is called the pre output block in our sixteen round cipher system.

The permutation $IP^{\wedge(-1)}$ applied to the pre output block is the inverse of the initial permutation IP applied to the input

(plaintext). Therefore, the pre output is then subjected to the permutation IP $^{(-1)}$ according to Table-8, which is given below.

Table-8 Inverse of Initial Permutation, IP $^{(-1)}$

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	.4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

Now Y = IP $^{(-1)}$

= (y_1, y_2, y_3,……y_{64})

= (1000 0101 1110 1000 0001 0011 0101 0100
0000 1111 0000 1010 1011 0100 0000 0101)

= (8 5 E 8 1 3 5 4 0 F 0 A B 4 0 5)

Which completes the sixteen round cipher text computation.

3.6.5.3 Decipherment: For the purpose of deciphering, it is necessary to apply the same DES algorithm as used for encipherment, to an enciphered data block. In each round of computation, the same key bits K_i are used during' decipherment. During a deciphering operation, Kj_6 must be used in round one, K_{15} in round two, and so forth.

The 64 bits of the cipher text Y to be deciphered are first subjected to the initial permutation IP of Table-4, and so on

CHAPTER 4

Image processing

4.1 Overview: Image processing can be defined as the techniques that require modification or interpreting of the existing pictures. In order to apply Image processing methods, the first step is to digitize the photograph or other picture into an image file. The digital methods can be applied to rearrange picture parts, to enhance colors separations, or to improve the quality of shading. These techniques are used extensively in commercial art applications that involve the retouching and rearranging of sections of photographs and other artwork. An example of the application of image processing methods to enhance the quality of a picture is shown in the figures below.

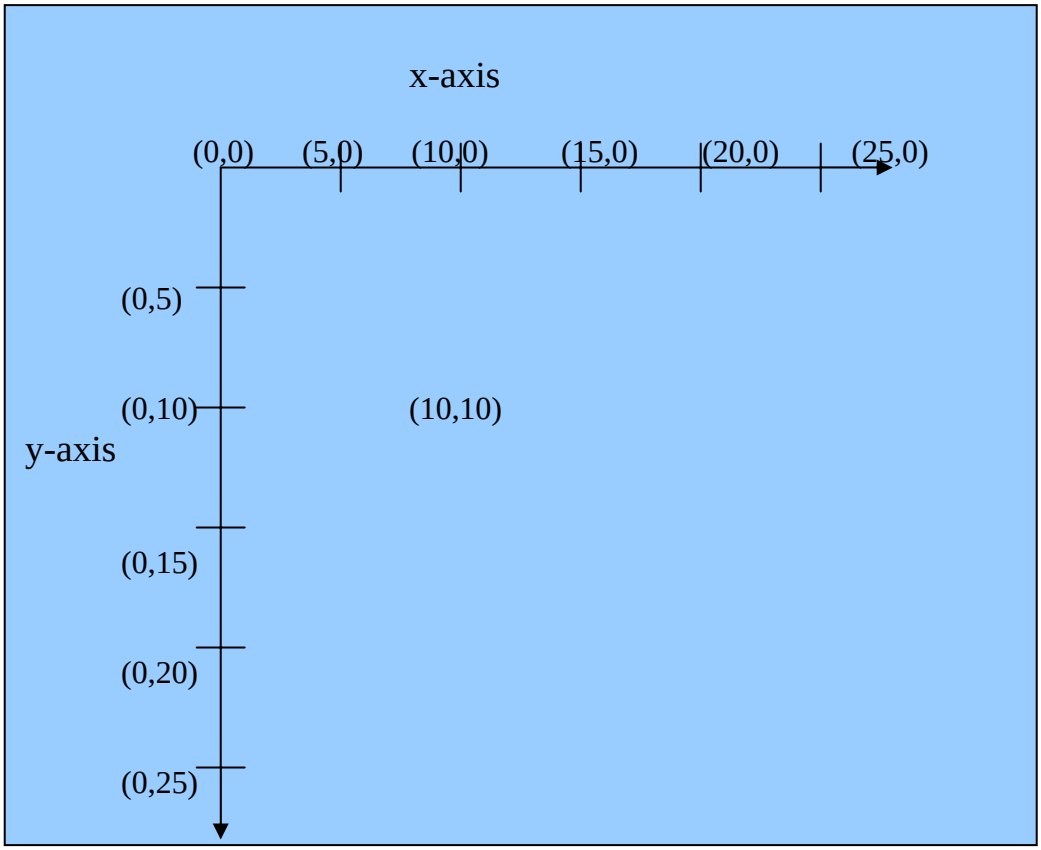
Picture quality before applying Image processing techniques

Picture quality after applying Image processing techniques

4.1.1 Pixel: Picture definition is stored in the form of screen points. Each screen point is referred as a pixel or pel (shortened form of picture element). Intensity range for pixel position depends on the quality of the system on which it is being displayed. In a simple black and white system, each screen point is either on or off, so only one bit per pixel is needed to control the intensity of the screen positions. Additional bits are required when color and intensity variations can be displayed. Generally 24 bits per pixel ar4e included in high quality systems

4.1.1.1 Graphics coordinate system: The x and y coordinate parameters in all of the drawing methods of the Graphic class apply to the Graphic Coordinate System. For an explanation of how the x and y

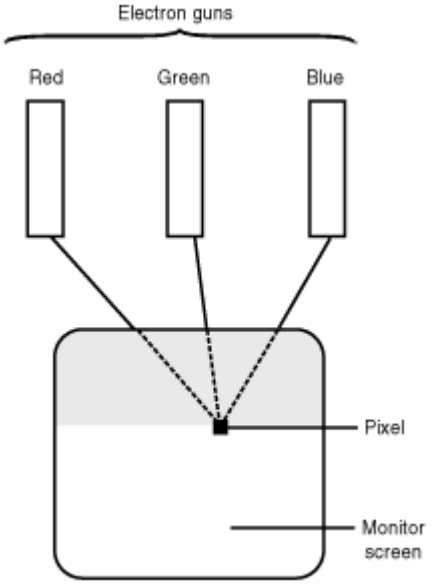
coordinates of an object about to be drawn relate to user's display area. This thing is explain in the figure below:



4.2 **Understanding Color**: Before jumping into the specifics of what a color model is, it's important to understand how color is represented on a computer in general. Although most operating systems have some degree of platform-dependent handling of color, they all share a common approach to the general representation of colors. Knowing that all data in a computer is ultimately stored in a binary form, it stands to reason that physical colors are somehow mapped to binary values (numbers) in the computer domain. The question is, how are colors mapped to numbers?

One way to come up with numeric representations of colors would be to start at one end of the color spectrum and assign numbers to each color until you reach the other end. This approach solves the problem of representing a color as a number, but it doesn't provide any way to handle the mixing of colors. A computer color system needs to be able to handle mixing colors with accurate, predictable results.

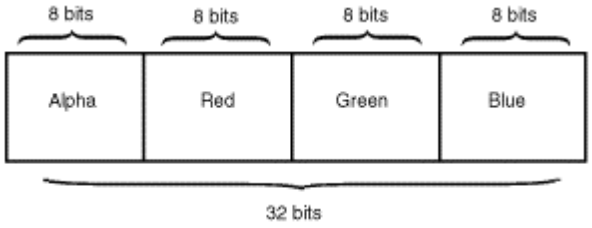
The best place to look for a solution to the color problem is a color computer monitor. A color monitor has three electron guns: red, green, and blue. The output from these three guns converge on each pixel of the screen, exciting phosphors to produce the appropriate color as shown in the figure. The combined intensities of each gun determine the resulting pixel color. The monitors use only these three colors (red, green, and blue) to come up with every possible color that can be represented on a computer.



Knowing that monitors form unique colors by using varying intensities of the colors red, green, and blue, you might be thinking that a good solution to the color problem would be to provide an intensity value for each of these primary colors. This is exactly how computers model color. Computers represent different colors by combining the numeric intensities of the primary colors red, green, and blue. This color system is known as RGB (Red Green Blue) and is fully supported by Java.

4.3 **Color Images in Java:** Bitmapped computer images are composed of pixels that describe the colors at each location of an image. Each pixel in an image has a unique color that is usually described using the RGB color system. Java provides support for working with 32-bit images, which means that each pixel in an image is described as using 32 bits. The red, green, and blue components of a pixel's color are stored in these 32 bits, along with an alpha component. The alpha component of a pixel refers to the transparency or opaqueness of the pixel.

A 32-bit Java image pixel is therefore composed of red, green, blue, and alpha components. By default, these four components are packed into a 32-bit pixel value, as shown in Figure . Notice that each component is described by 8 bits, yielding possible values between 0 and 255 for each. These components are packed into the 32-bit pixel value from high-order bits to low-order bits in the following order: alpha, red, green, and blue. It is possible for the pixel components to be packed differently, but this is the default pixel storage method used in Java.



Representation of a pixel in java

A color component value of 0 means the component is absent, and a value of 255 means it has its maximum intensity or value. If all three color components are 0, the resulting pixel color is black. Likewise, if all three

components are 255, the color is white. If the red component is 255 and the others are 0, the resulting color is pure red.

The alpha component describes the transparency of a pixel, independent of the color components. An alpha value of 0 means a pixel is completely transparent (invisible), and an alpha value of 255 means a pixel is completely opaque. Values between 0 and 255 enable the background color to show through a pixel in varying degrees.

4.4 Bitmap Images: The simplest bit image type is the bitmap image, which is a pixel image using 2, 16, or 256 colors. Individually bitmaps may be as large as the full screen, medium sized as in the solitaire card images, or as small as a few dozen pixels for a check box control or radio button.

Bitmap images can be created by any paint program and do not differ in any respect from conventional bitmaps. This means that bitmap images can be imported from any external sources.

4.4.1 Device Independent Bitmaps: The device independent bitmap (DIB) format originally appeared as an extension of OS/2 Presentation bitmap format. This format presents, as its most important feature, an RGB color table defining all colors used in the bitmap. Most of the bitmap editors or paint automatically create DIB image files. Because DIB bitmaps have become so common, the extension .DIB is rarely used; files bearing the .bmp extension are almost always device independent images.

4.4.1.1 DIB File Format: The DIB image file format consists of several sections: the Bitmap Info header, the color table and the image data. Each of these is described as under:

4.4.1.1.1 The Bitmap INFOHEADER: The bitmap INFOHEADER provides information about the structure of the bitmap itself. The bitmap INFOHEADER consists of the 54-byte record shown in table as under:

BITMAPINFOHEADER Data

Field	Size	Sample	Val	Description
biSize	DWORD	28000000	28h	Size of Bitmap INFOHEADER
biWidth	LONG	08000000	8h	Bitmap pixel width
biHeight	LONG	08000000	8h	Bitmap pixel height

biPlanes	WORD	0100		1h
		Color planes (always 1)		
biBitCount	WORD	0400	4h	Color bits per pixel (1,4,16,24)
biSizeImage	DWORD	20000000	20h	Bitmap size for compression

4.4.1.1.2 **The Bitmap Color Table:** This table consists of a series of RGBQUAD structures. These are read, in order, with the first byte blue, the second byte green, the third byte red, and the fourth byte in each QUAD set to zero. The number of RGBQUAD structures is identified by the biBitCount field. If biCount is 8, 256 RGBQUAD values are required.

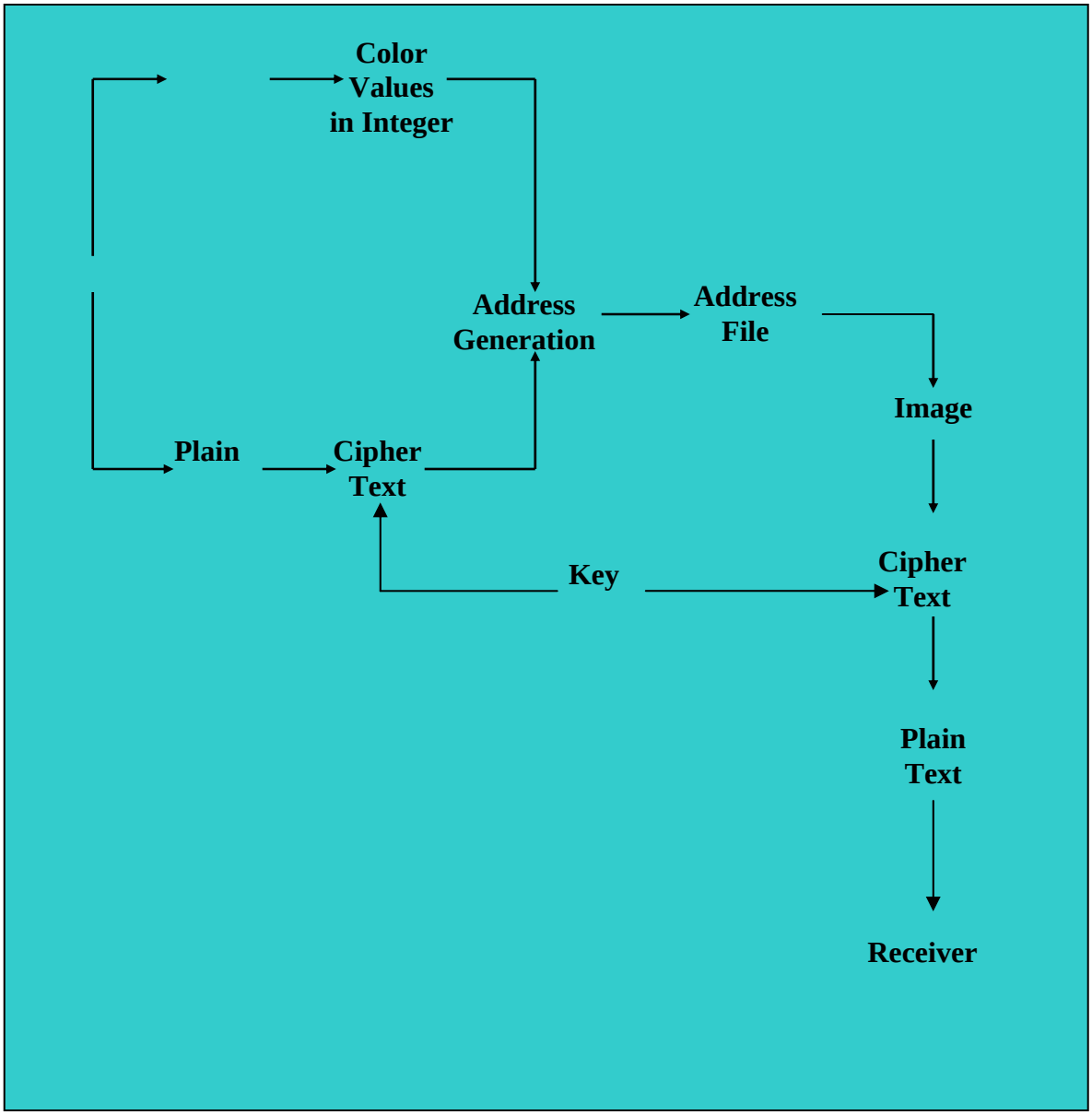
4.4.1.1.3 **The Bitmap Image:** The final section of the bitmap file is image itself. The arrangement of this section mainly depends on the number of colors, but it is also affected by two other factors, which are constant for all bitmaps.

- Each row of the bitmap image must be even multiple of four bytes. Each data row begins with the left most pixel of the scan line and is right padded with zeros as necessary.
- Unlike the original bitmap format, the today's bitmap format for DIBs begin with the bottom scan line in the image, not the top.

CHAPTER 5

PROJECT IMPLEMENTATION

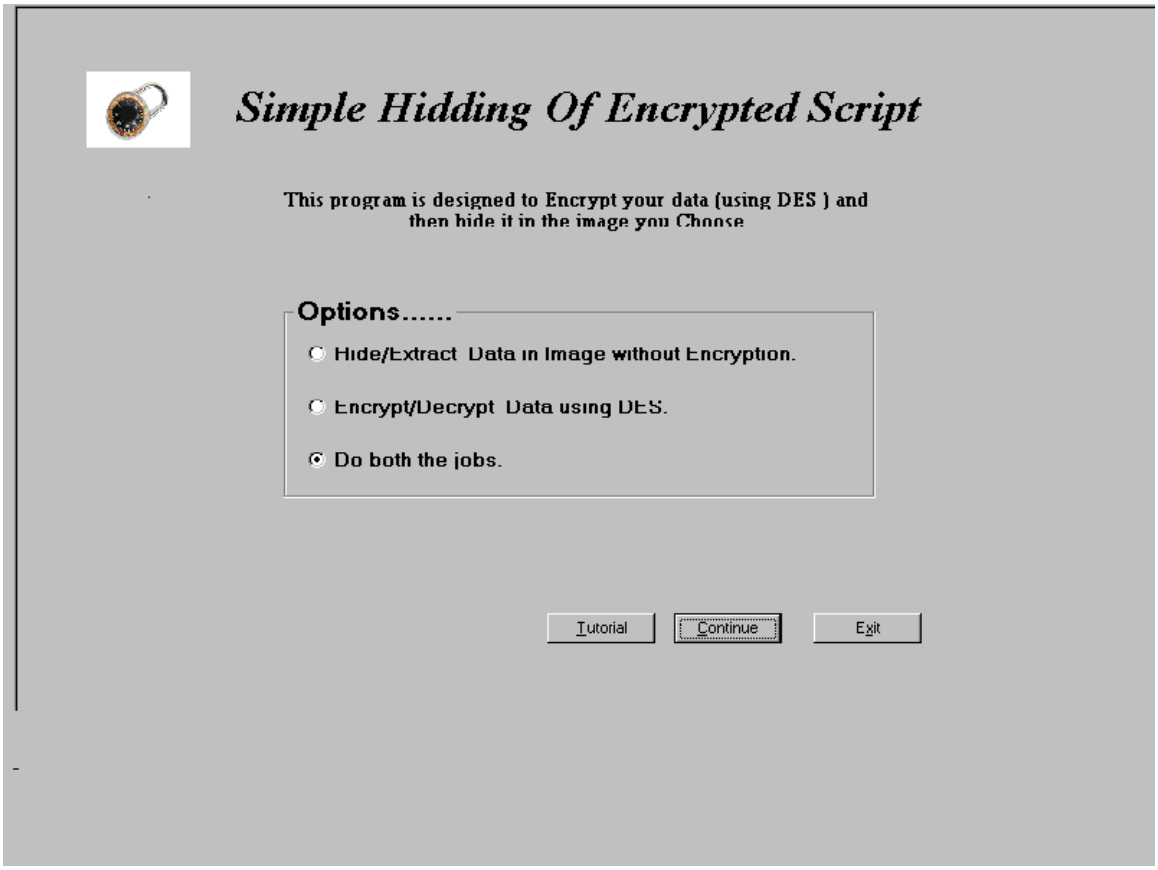
5.1 Conceptual Model:



The sender will provide two inputs to the software. One will be the plain text to be encrypted and the other input will be in the form of a bitmap Image. The plain text needs to be written in the **Notepad** or **WordPad** with the file having .txt extension. The DES module of the software will encrypt the plain text and will give output in the form of **cipher text**. The image will be digitized and will be broken into **pixel** form. Each pixel of the image will further be broken into its individual color components. The intensity value of the each color component in each pixel will be calculated. The ASCII codes of each character of cipher text will calculated and will be matched with the individual color value. The look ahead for match will continue till the ASCII value of cipher text finds the equivalent color value. When the match is found, the address of the color component within the image will be stored in the **address file** that in turn will be transferred to the receiver. The address file will be written in the form of **Long** data type that will be unreadable for the intruder. This address file will be transmitted to the receiver who will find the ASCII codes of the cipher text by making use of address file received from the sender. Once the cipher text is obtained than the reverse DES algorithm will be applied and the plain text will be obtained. Thus completing the complete process of encryption and decryption.

5.2 Front End: The front end of the project is developed in Microsoft Visual j++. Visual j++ provides Microsoft's GUI based method of developing, compiling, testing, and debugging java programs. One of the major benefits of Visual j++6.0 is that it uses the same basic development environment that experienced visual C++ programmers have mastered, with some minor modifications. Additionally, like other visual programming languages, Visual j++ offers many auxiliary applications that simplify some of the more common and/or difficult tasks associated with Java program development. Efforts have been made to keep the front end much user friendly. As the windows is the operating system installed on most of the computers in the market today, so special emphasis has been given to keep it close to the window environment. Thus anybody knowing little bit of windows environment will feel very comfortable with the software.

5.2.1 **Welcome window:** This is the first window that the user will find once it runs the software.



This window performs more than one function. The first thing it does is that it provides choice of performing the job. Three types of radio buttons are provided to enable the user to get the appropriate type of job from the software. The choices are:

- It allows the user to encrypt the plaintext by making use of DES only.
- It allows the user to encrypt the data by making use of DES and than enhancing the security level by hiding the data in the Image.
- It allows the user to hide the data in the Image without making use of DES.

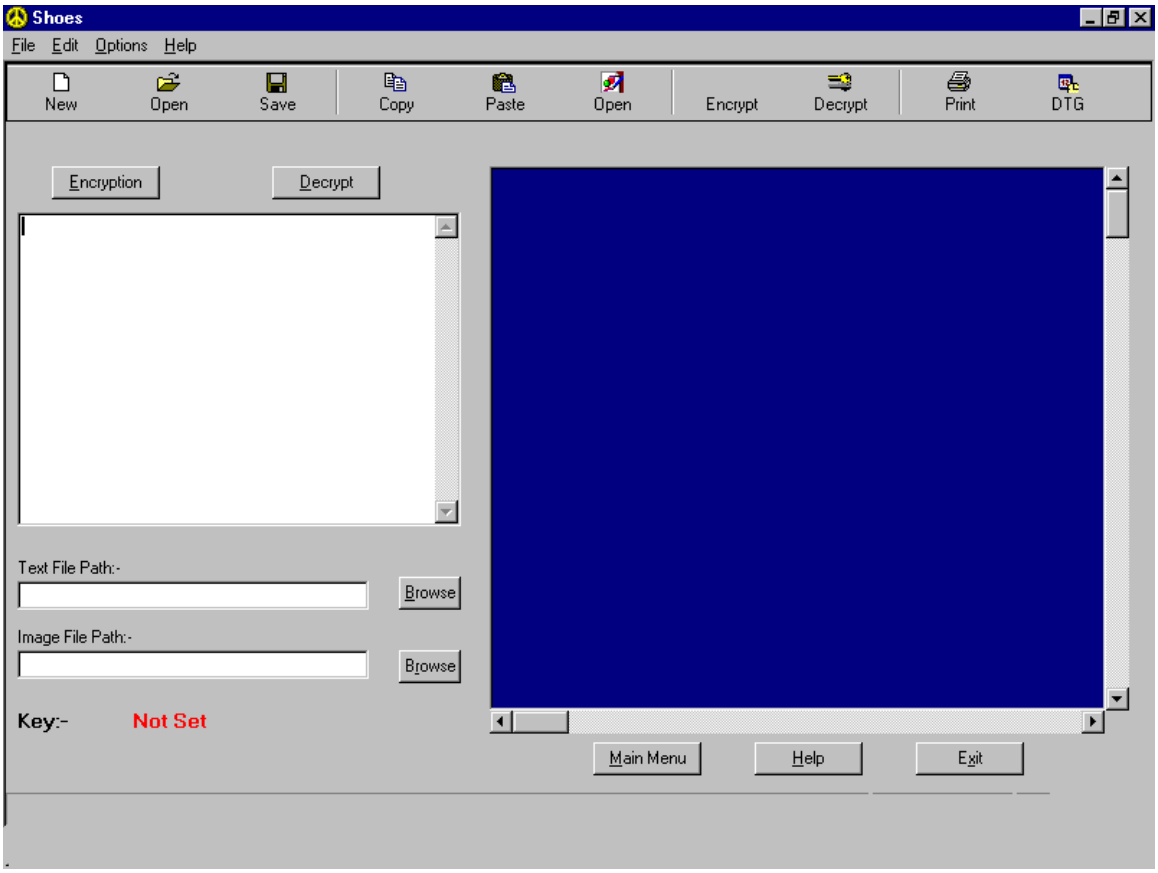
The second thing it does is that it allows the user to go through

the tutorial, which might be required by the user new to the software by clicking the **Tutorial** button .

The third and last thing it allows user to terminate the project and exit immediately by clicking the **Exit** button.

Finally the **Continue** button allows the user to start performing the selected task by moving to next part of the software.

5.2.2 **Main Window:** This is the next window that user will find once it has clicked the continue button of the welcome window. The windows looks like as under:



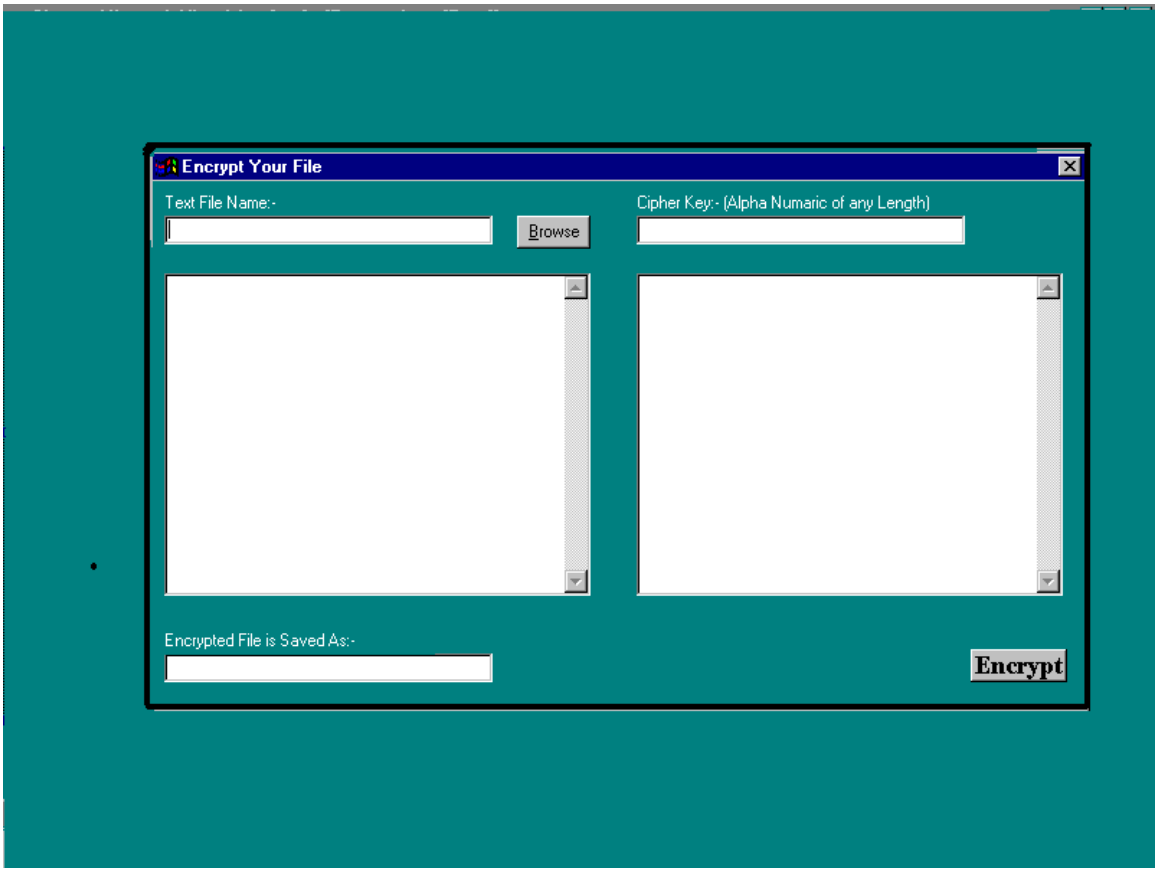
It allows the user to perform number of functions that are as under mentioned:

- The first job that the user must perform is to decide for the type of job for the software to perform. User must decide whether he wants to perform encryption or decryption job.
- If user selects the encryption job for the software to perform than he must give following inputs to the software:
  - User must provide the text file path that he wants to encrypt.

- User must provide the Image file path that he wants to use for hiding the data.
- Once the above two inputs are provided than the user must click the **Encryption** button to proceed to next part of the software.

- If user selects the encryption job for the software to perform than he must give following inputs to the software:
  - User must provide the address file path to retrieve the cipher text from the Image.
  - User must provide the Image file path that he wants to use for retrieving the cipher text.
- Once the above two inputs are provided than the user must click the **Decryption** button to proceed to next part of the software.
- User has the option to include the date time stamp in to the text portion in case of encryption.
- It also allows the user to switch for any help if required by just clicking the **Help** button.
- It also allows the user to switch back to Welcome window by clicking the **Menu** button.
- It also allows the basic text functions to be executed like **Cut, Paste, Word wrap** and **Exit**.
- It also allows the user to go through the tutorial of the software.
- It also allows the user to know about the developers of this software and the supervisor who supervised this project.

5.2.3 **Encryption Window:** This is the next window user will find once it has clicked the encryption button after giving all required inputs in the main window. The window is shown on the next page.

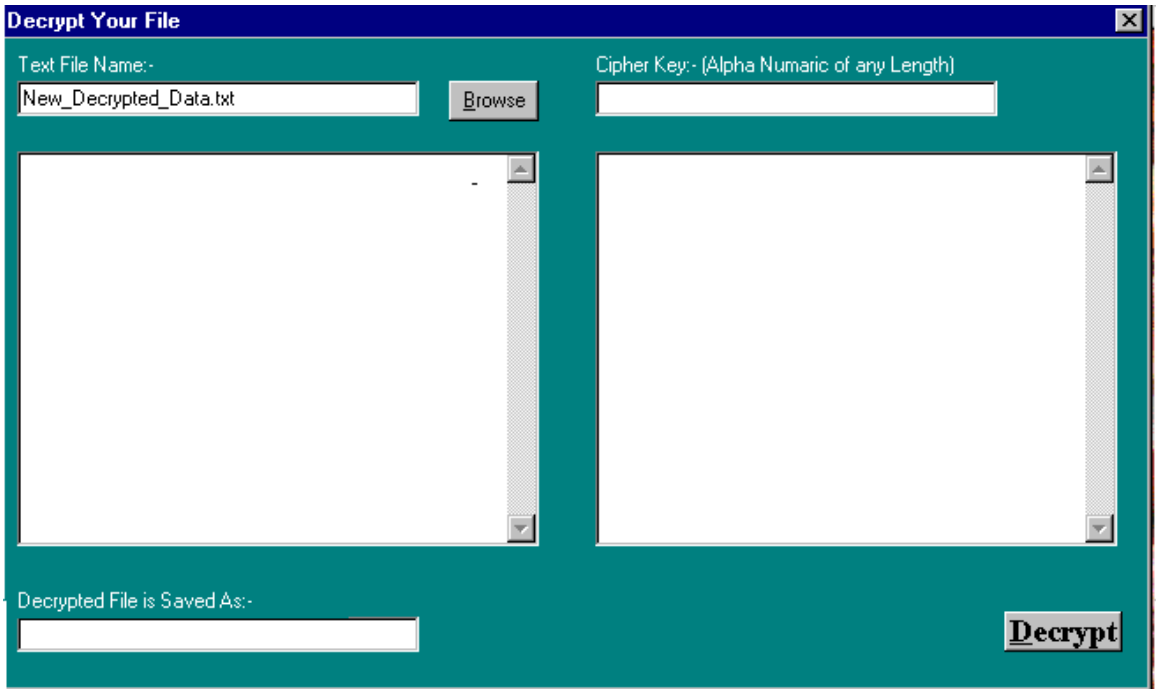

Encrypt Your File
Text File Name:-
Browse
Cipher Key:- (Alpha Numaric of any Length)
Encrypted File is Saved As:-
Encrypt

In this window user will give the key required for the purpose of encryption. The user will also give the path for saving the encrypted file in case encryption is done without data hiding in the image.

Once all the above requirements are fulfilled than user must click **Encrypt** button. The user will get the result message displaying all the information user has provided so far once he clicks the Encrypt button. The **result** window is shown below:

The result window confirms that encryption process was completed with success. It also reminds the parameters required for the decryption process. The encryption part finishes with click of the **Ok** button of the result window by the user.

5.2.4 <u>Decryption Window</u>: This is the next window user will find once it has clicked the decryption button after giving all required inputs in the main window. The window is shown as under:



In this window user will give the key required for the purpose of decryption. The user will also give the path for saving the encrypted file in case decryption is done without data hiding in the image.

Once all the above requirements are fulfilled than user must click **Decrypt** button. The user will get the result message displaying all the information user has provided so far once he clicks the Decrypt button.

5.3 <u>Data Hiding in Image</u>: is done to enhance the security level of the software. The chances of breaking of DES has increase manifold due to the presence of faster computing machines. Thus to enhance the security of the data while making use of DES we have made use of Image processing techniques. The complete implementation of this module of the software can be explained under following sections:

5.3.1 Loading of Image: The first and the foremost thing for performing Image processing techniques are to load the Image. Java does provide a very convenient method to load an image by making use of Picture Box's get Image() method. The disadvantage of this technique is that the image read from Picture Box becomes hard coded. Thus user has to have relied on the hard coded image offered by the Picture Box. As the Image in our project is used for the purpose of security, so there was a requirement of using more than one Image in the software. Secondly by using just one image the chances of compromising of image were quite high. That is the reason we have not used any built in classes provided either by Sun Microsystems or Microsoft. The second reason for not using any built in classes for loading of the image is that both the image classes o f Sun Microsystems and Microsoft are not compatible with each other.

The solution to this problem was that to read the image as a stream and then perform the requisite job on it. So we made use of java IO classes to read the image as a stream. This method also allowed the software to use any type of bitmap format image. So we have given the choice to the user to provide any image to the software with following exceptions:

- The image provided must be of the .bmp extension.
- The image must have all color values in the range –127to +128 in order to cater for all 256 ASCII codes.

5.3.2 Getting Pixel Data From Image: After the image is loaded, one must calculate the number of Pixels in the image. For this one has to find the **width and height** of the image. Once the width and height is known the total number of pixels can be easily found by just multiplying the width by the height. The next thing we were required to do was that to create a buffer large enough to hold the entire image data. At this moment we have to cater for following points:

- As we have used the IO classes to read the image so the image will be stored in the form of array of bytes instead of array of integers, which is usually the case.
- Data will be stored in the byte array will be an exact duplicate of the image's disk file. As we have already stated previously that BMP images when they are stored on the disk, have two headers before the actual image data. Those two headers will be stored in the first part of the byte array. The total size of both of

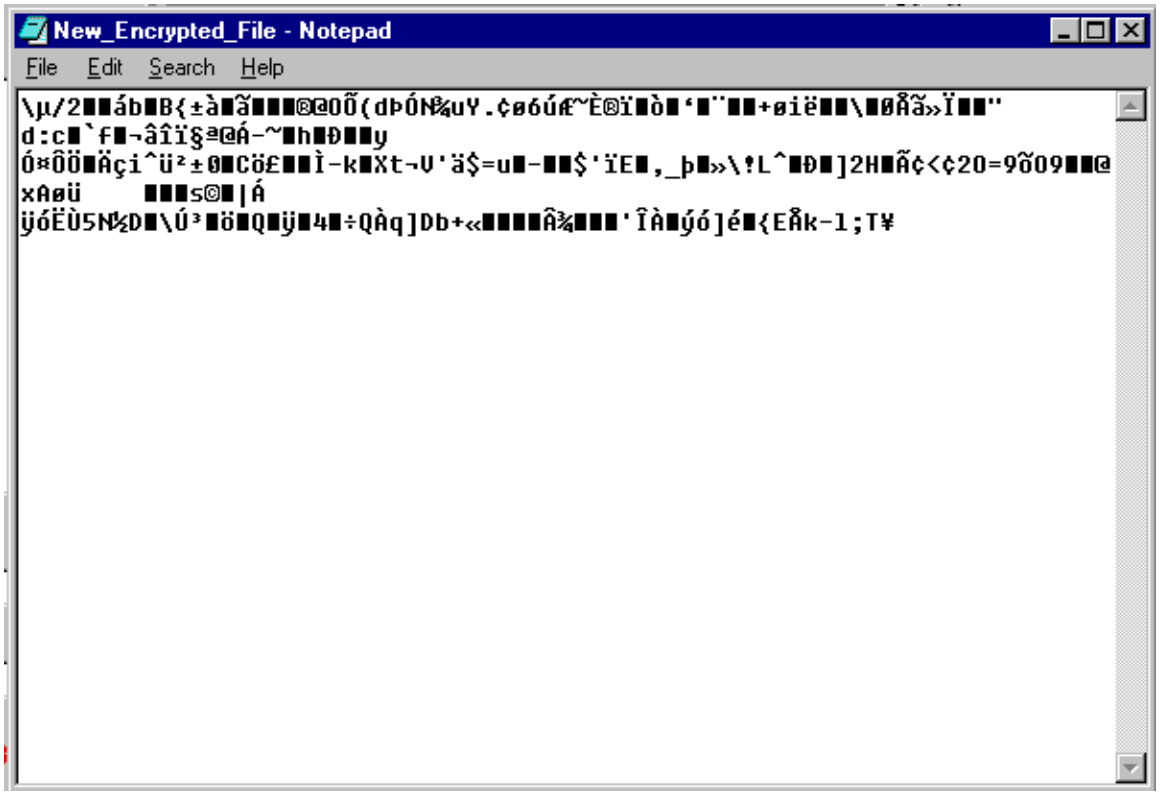
the BMP headers is 4 bytes. So we have to allocate the space for this byte header as well while allocating space for the buffer.

- As we have already mentioned that this is a byte buffer and the pixel that's contained in the BMP image files has three bytes per pixel describing the red, green and blue components of the pixel. So we have to multiply the width of the image by 3 in order to get the actual size of the image's byte width.
- Still there is one more twist in the form of BMP files. Every scan of a BMP file must be an even multiple of 4 bytes. Let's say we have a scan line of two pixels. Each pixel is of 3 bytes. That gives us total 6 bytes for the scan line–but that's not an even multiple of four . the file will actually contain 8 bytes for the scan line. The last 2 bytes will be completely ignored, but still we need to allocate space so that the entire scan line of data will be stored.

So we have to make all above-mentioned checks before allocating any space for the buffer to hold the entire image data.

5.3.3 <u>Getting ASCII Codes Of the Cipher Text/Plain Text</u>: The next thing that is done in this module is to calculate the ASCII codes of each character of the Cipher text or plain text depending on the type of the input provided to the module. The input provided is read character by character and its equivalent ASCII codes are calculated and stored in an array. As the color component value of the image ranges in between –127 to +128 and the ASCII values ranges in between 0-255, so 256 has to be subtracted from any ASCII value which is greater than 127. this is done in order to bring the ASCII values within the range of color component values of the image.

The form of cipher text usually received from the DES module of the software is shown in the figure below.



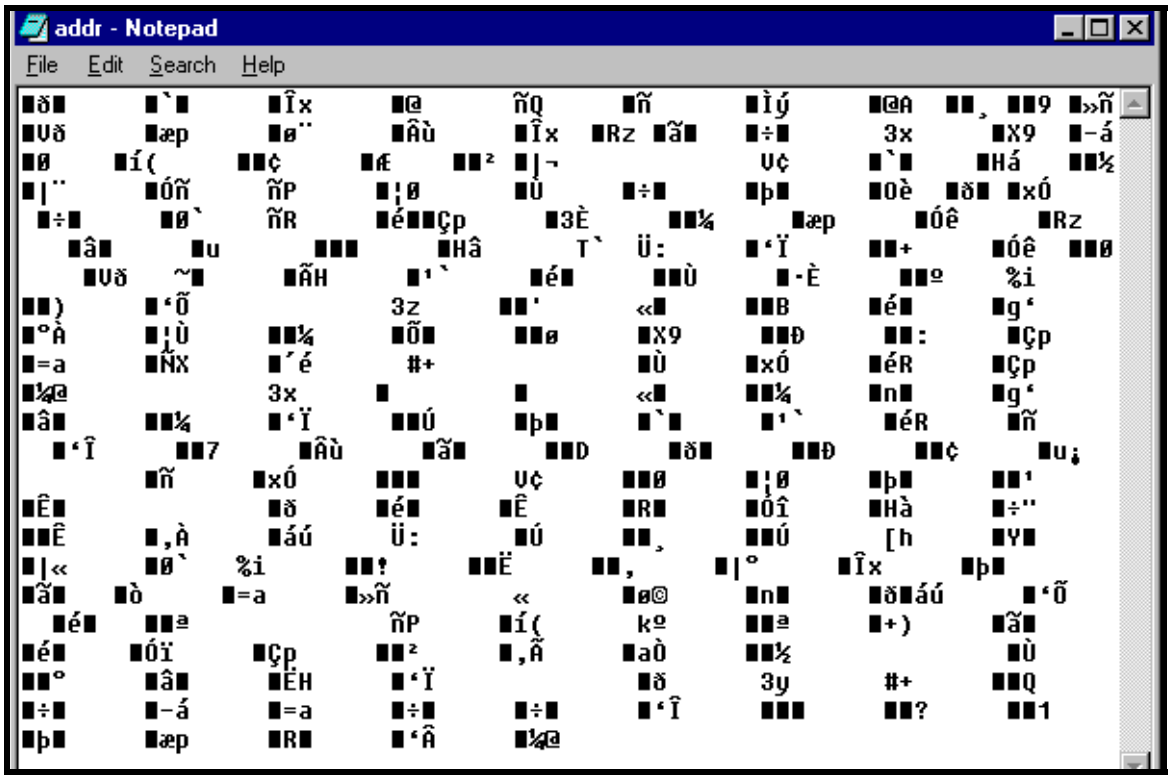
Ciphered Text Received From DES Module

5.3.4 Getting Color Component Value of the Image: by now we have calculated the number of pixels in the image and have created the

buffer for holding the entire image. Each pixel is formed of three colors that are stored in the byte form. The bytes go in the order of the blue byte, green byte and finally the red byte.

The first thing we do is that we find the offset of the pixel that is found by keeping the x coordinate as 0 and scanning from the max value of y coordinate. As we have already mentioned that BMP images are scanned from bottom to top. Offset is simply the starting address of each pixel. We found the value of individual color component value by adding the pixel-offset value with the color-offset value. The color-offset values are 2,1and 0 respectively for red, green and blue component of the pixel. Finally as the values returned are in the form of byte and ASCII values are in the form of integers, so type casting is done to find the equivalent integer values of each color component.

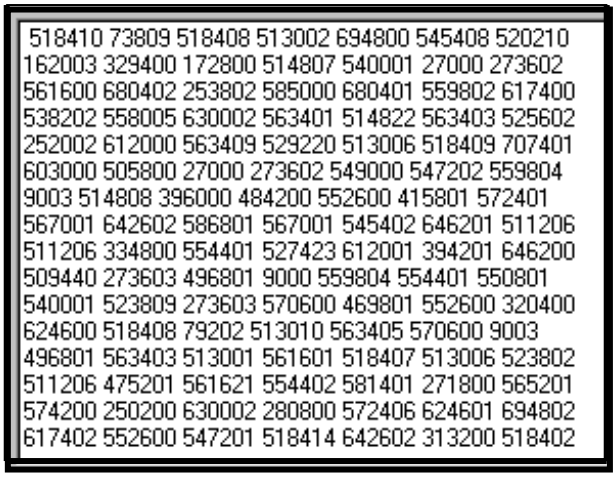
5.3.5 Getting Address File: Once we have got the integer values for both the color components and the cipher/plain text. The next step is to find the match between the two values. We start with the ASCII code of each character and match it with the color component value. We keep on scanning each pixel one at a time till we found the match. Once the match is found the address of the matched color in the image is calculated. That is done by making use of the pixel-offset and then adding the color-offset in it in order to find the exact address of the specific color in the image. The addresses are stored in an array of long data type. This is done because to cater for the values of addresses which goes out of the range of integer data type. Once all the characters of the cipher/plain text have been matched than this array is written on a file. The addresses are also written in long format that provides extra security to the data because it is unreadable for a person opening it without the help of this software. An address file that goes to the destination is shown in the figure on next page. Although it will be different for different texts but its look will be primarily be the same.



As one can see from the above figure that the contents are unreadable till thee time one uses the same software to open it.

5.3.6 <u>Writing Of Image</u>: Although for the normal practices, the will not be required to be saved again on the disk but our software provides the flexibility to resave it on the disk. Before writing the image on the disk, we require to save it again in the same format as it is saved on the disk. After performing all the jobs on the image, we calculate the byte width for each scan line and store it. As we already know that the BMP file data is stored in upside down so we simply transform the y coordinate. We again calculate the pixel-offset and finally we set the three colors components in the data buffer declared previously. this process is repeated for each pixel-offset till we have covered each pixel of the image. Finally we again write the image as a stream object on the disk. Here again out GUI asks the user to provide the path for file where the image has to be saved. Encryption portion of our software finishes at this stage and user can now simply send the address file to the destination end and if required the image as well. But one thing user must keep in mind that he must not send the both files simultaneously and on the same communication channel.

5.3.7 <u>Reading of Address File</u>: The address file is read in the same manner as it is written. Once the file is read, it is stored in an array of long data type. Here one thing has to be kept in mind, file should be read with the same IO classes as it is written. If both of them are non-compatible than the garbage values will be read. If properly read than the address file will look something like similar as shown in the figure below. These are the addresses of the color components.



<u>Addresses of the Cipher/plain Text</u>

The thing notice in the above figure is that there are addresses

which appear more than once. So by repeating the addresses again we can have the flexibility to use file of any size for hiding the data. The size of the file may be larger than the size of the image itself. Only thing that must be kept in mind is that image must have all 256 color values. If that is true than size of the file to be hid does not matter.

5.3.8 Extraction Of Cipher/Plain Text: After we have read the address file, we read the image in the same manner as we have done it for encryption process. Then we calculate the number of pixels in the same manner as we did in case of encryption. As there is no need of the image after the decryption process is complete, so we need not to create any source buffer to hold the image data. Once the image is loaded than the next thing is to perform extract the color values from the specified addresses. These values are stored in an integer array. As there are some negative values as well, so values less than zero are converted into positive by adding 255 to them. Thus making them in the range of 0-255, which is the range of the ASCII codes. Once the values are made positive than the equivalent characters for those ASCII codes are found and stored in a char array. If the decryption is done without Des than the output of this module will be the plain text else it will be the cipher text. Whatsoever be the case the output will be stored in a file on the disk and will be handled accordingly.

5.4 Creating a Self-Extracting Setup Distribution Project: The most common method for distributing the application is self-extracting setup program. Almost all software packages today contain such a program, which must be run in order to install anew application. When the user will execute the program, he will receive all of the necessary files for running the application.

The main benefits of the self-extracting setup program is that the end user needs to know a very little about the application and its installation. Additionally, the user can use the program's options to ensure that the application and its components are installed where and how you expect, thus reducing potential maintenance headaches in the future. The need to create self-extracting our project as self-extracting project is primarily due to the nature of the usage of our application. Following are the steps that were followed to create a self-extracting setup distribution project.

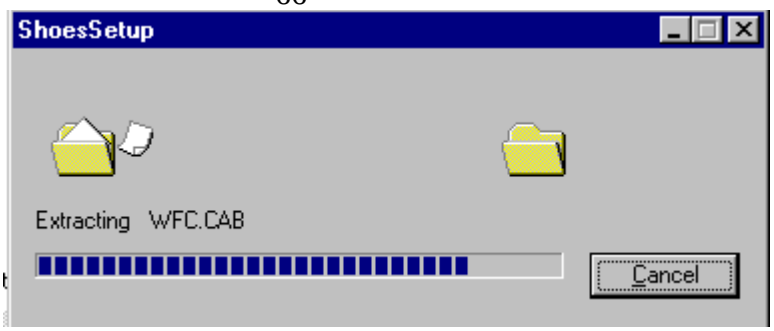
- The first and foremost step is to place the distribution project within the same solution as the project itself. Here one must make sure that he should not place the distribution project in the same directory as the development project, because if that is done than the next

programmer that modifies this application will have access to both the projects

- Once the self-extracting setup project is added to the developed project, the next step will be to add the output of the developed project to the self-extracting project.
- Visual java does provide options to the developer to give the default path for the place where the project should get installed on the computer after the installation process get completed successfully.
- As the many of the user who will install this program will not have the WFC libraries and Java virtual machine, which are required for proper working of application on their PCs. So we have given the flexibility to the user that he will get these installed on his computer along with our application when he installs our software on his computer system. In this way our software becomes completely platform independent as java virtual machine will be installed on the target PC by our software and user will not require any java compiler to run this software on his PC.
- Various windows and their functions which user will come across during the installation process are as under.
  - User will just have to click the icon shown in the figure below to start the installation process. This icon will be present on the CD Rom provided with our software.

In the above figure, one can see the highlighted icon with the name of shoes setup. Whenever the user will click this icon the setup process will start.

  - Once the icon is clicked the software will start extracting the required file itself as shown in the figure below.

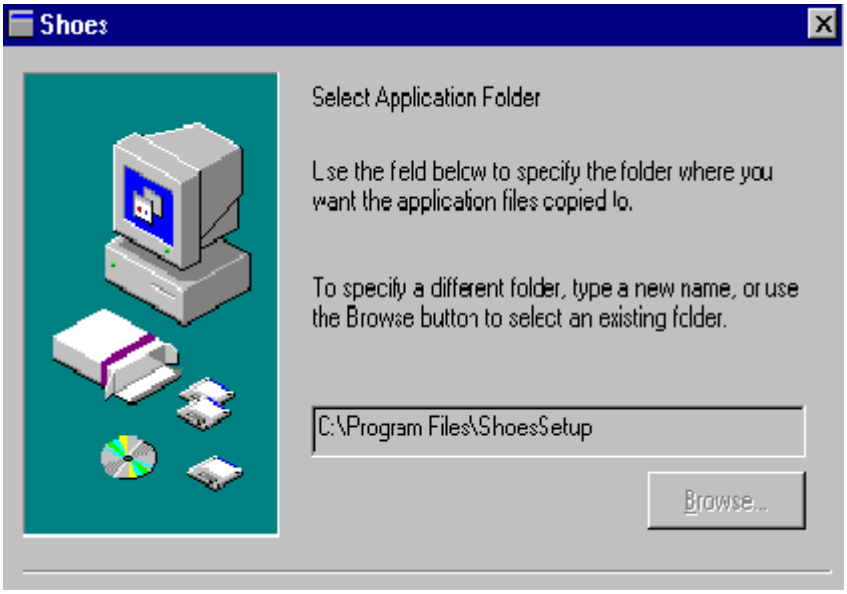


In the above figure, one can see the message written of "Extracting WFC.CAB".

- Once the software completes extracting the require files from the CD Rom than the next window will be as under. The next window will be welcome window, which will ask the user to close down any programs already opened.

- It will also inform the user that software is about to install the necessary files on the PC.

- Once the user clicks the **Continue** button on the next window than in the next window, software will prompt from the user to provide the path where he wants to install the software. The window is shown below.



- Finally once the path is also provided than the installation will take place. Software has the ability to create the path, if the path does not exist already. Once the installation will

get complete, the user will be informed about the completion of the installation.

5.5 C lasses Used: Complete list of classes used by our project are given as annexure A.

5.6 Complete Code: Complete code of the software develop is given as Annexure B.