CRYPTOGRAPHY FOR IMAGE COMMUNICATION

By

ALEM HADDUSH
BETHELIHEM TESFAYE, &
MEHARI LBELO

Advisor
Ato ESUBALEW ADAM

A project report submitted to the

ELECTRICAL ENGINEERING DEPARTMENT

in partial fulfillment of the requirements
for the award of the degree of

Bachelor of Science
in
Electrical Engineering

ELECTRICAL ENGINEERING DEPARTMENT
FACULTY OF ENGINEERING
BAHIR DAR UNIVERSITY
BAHIR DAR

June, 2006

CERTIFICATE

Certified that this project entitled "Cryptography for image communication" is the work of

ALEM HADDUSH,
BETHELIHEM TESFAYE, &
MEHARI LBELO

who carried out the project under my supervision. Certified further that to the best of my knowledge the work of the reported herein does not form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

Ato Tefera Abate (MSc)
Head of the Department
Department of Electrical Engineering
Faculty of Engineering
Bahir Dar University
Bahir Dar

Ato Esubalew Adam
Advisor
Department of Electrical Engineering
Faculty of Engineering
Bahir Dar University
Bahir Dar

Acknowledgement

We would like to articulate our appreciation and special words of thanks to all people who have contributed to the successful completion of our project. Most specially, we would like to extend our indebted gratitude to our advisor Ato Esubalew Adam for his unending support and advice right from the beginning to the end without which our project would be incomplete. We finally take the opportunity to thank our department, EE, for providing us with materials vital for our work.

List of figures **Page**

Fig 1.1. Encryption and decryption………………………………………………..…1

Fig1. 2. Three types of cryptography……………………………………….……...3

Fig 1.3 Symmetric Cryptographic Model ……………………………………...….3

Fig 1.4 Asymmetric Cryptographic Model……………………………..………4

Fig 1.5 passive attack…………………………………………………………....5

Fig 1.6 Active attack……………………………………………………………6

Fig 2.1 Encryption and decryption……………………………………………....12

Fig 2.2 The key-schedule of DES …………………………………………….……13

Fig 2.3 The overall Feistel structure of DES……………………………………..17
Fig 2.4 The Feistel function (F-function) of DES……………………...……...18
Fig 3.1 Asymmetric encryption………………………………………………..23
Fig3.2 Sending secure messages with public key encryption…………………...24
Fig.4.1 Signature Generation…………………………………………………..33
Fig.4.2 Signature Verification…………………………………………………33
Fig 4.3 Message digests generation using SHA-1……………………………..…35
Fig 4.4 Processing of a single 512 –Bit Block……………………………….…...36
Fig 4.5 SHA-1 single step processing block ………………………………...…...37
Fig 5.1 Two-prime number generation……………………………….……….……39
Fig 5.2 RSA key generation ………………………………………………...…40
Fig 5.3 DES key encryption…………………………………………….……....41
Fig 5.4 DES key decryption………………………………………………..……..41
Fig 5.5 input image……………………………………………………………..42
Fig 5.6 gray scale input image…………………………………………………...…42
Fig 5.7 resized gray scale image…………………………………………….....…42
Fig 5.8 matrix of the pixel values of the resized gray scale original image…..…..45
Fig 5.9 matrix of the pixel values of the encrypted image……………………..…47
Fig 5.10encrypted image……………………………………………………......48
Fig 5.11 decrypted pixel…………………………………………………..…49
Fig 5.12 decrypted image…………………………………………….….…..…49
Fig 5.13 message digest of the original image……………………………..……...50
Fig 5.14 encryption of the message digests of the original gray scale image……..50
Fig 5.15 Message digest of the decrypted image……………………………......51
Fig 5.16 comparison of the message digests of unmodified image………………..51
Fig 5.17 Message displayed if the image is hacked during transmission…………52

Abstract

This project focuses on cryptography for image communication. That's, it deals with the encryption and decryption of images for secure communication. Furthermore, it explains how digital signatures are applied to images to enhance security and to verify authentication. Basically, cryptography is the art and science of keeping messages (image) secure by hiding them or by transforming them into a form which is unintelligible during transmission over communication channels (networks).

In a broad sense, based on the number of keys employed for the processes of encryption and decryption of image and their functions, cryptographic algorithms are classified into three fundamental types, namely conventional (or symmetrical), asymmetrical (or public key), and hush function cryptographic algorithms. Each one of them is discussed in chapter and verse in the main body of the report. Encryption and decryption of images is implemented using the DES algorithm which takes a block of 64 bits and a key of any length as input. The key is randomly selected, but made so long as possible to make hacking (cryptanalysis) of the image so difficult.

Secure Hash Algorithm (SHA_1) is applied to generate digital signatures (or message digests). These digital signatures (or message digests) are used to authenticate the image, and to compare the transmitted and received images to check whether any form of attack has been launched on the image during transmission over the communication channel or not. Furthermore, digital signatures could also be created using RSA to authenticate the source and the transmitted image. Finally, sample outputs are produced for each and every process performed on the image.

Chapter-1
Introduction

Does increased security provide comfort to paranoid people? Or does security provide some very basic protections that we are naive to believe that we don't need? During this time when the Internet provides essential communication between tens of millions of people and is being increasingly used as a tool for commerce, security becomes a tremendously important issue to deal with. There are many aspects to security and many applications, ranging from secure commerce and payments to private communications and protecting passwords. One essential aspect for secure communications is that of cryptography, which is the prime focus of this work (project).

The origin of the word cryptology lies in ancient Greek. The word cryptology is made up of two components: "kryptos", which means hidden and "logos" which means word. Hence, cryptography is the science of writing in secret code and is an ancient art. Some experts argue that cryptography appeared spontaneously sometime after writing was invented; that's, the first documented use of cryptography in writing dates back to circa 1900 B.C. when an Egyptian scribe used non-standard hieroglyphs in an inscription., with applications ranging from diplomatic missives to war-time battle plans. In other words, cryptology is as old as writing itself, and has been used for thousands of years to safeguard military and diplomatic communications. For example, the famous Roman emperor Julius Caesar used a cipher to protect the messages to his troops. Within the field of cryptology one can see two separate divisions: cryptography and cryptanalysis. The cryptographer seeks methods to ensure the safety and security of conversations while the cryptanalyst tries to undo the former's work by breaking his systems. It is no surprise, then, that new forms of cryptography came soon after the widespread development of computer communications. In data and telecommunications, cryptography is necessary when communicating over any untrusted medium, which includes just about *any* network, particularly the Internet.

Cryptography comprises encryption and decryption. Image that can be read and understood without any special measures is called ***plaintext*** or ***clear image***. The method of disguising a plain image in such a way as to hide its substance is called ***encryption***. Encrypting plaintext results in unreadable gibberish called ***cipher text (noisy image)***. You use encryption to ensure that information is hidden from anyone for whom it is not intended, even those who can see the encrypted data. The process of reverting cipher text (noisy image) to its original plaintext (clear image) is called ***decryptio****n*. *Figure 1-1* illustrates this process.

Fig 1.1. Encryption and decryption

1.1 The purpose of cryptography

Within the context of any application-to-application communication, there are some specific security requirements, including:

> ***Authentication****:* The process of proving one's identity. (The primary forms of host-to-host authentication on the Internet today are name-based or address-based, both of which are notoriously weak.)
> ***Privacy/confidentiality****:* Ensuring that no one can read the message except the intended receiver.
> ***Integrity****:* Assuring the receiver that the received message has not been altered in any way from the original.
> ***Non-repudiation****:* A mechanism to prove that the sender really sent this message.
> ***Availability****:* It ensures and requires that the data are available to authorized parties

Cryptography, then, not only protects data from theft or alteration, but can also be used for user authentication. There are, in general, three types of cryptographic schemes typically used to accomplish these goals: secret key (or symmetric) cryptography, public-key (or asymmetric) cryptography, and hash functions, each of which is described below. In all cases, the initial unencrypted data is referred to as *plaintext (clear image)*. It is encrypted into *cipher text*, which will in turn (usually) be decrypted into usable plaintext.

In many of the descriptions below, two communicating parties will be referred to as Alice and Bob; this is the common nomenclature in the crypto field and literature to make it easier to identify the communicating parties. If there is a third or fourth party to the communication, they will be referred to as Carol and Dave. Mallory is a malicious party, Eve is an eavesdropper, and Trent is a trusted third party.

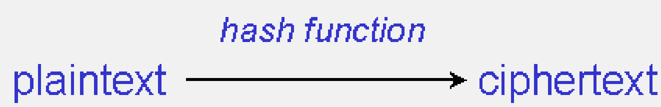
1.2. The different types of cryptographic algorithm

There are several ways of classifying cryptographic algorithms. However, for purposes of this paper (project), they will be categorized based on the number of keys that are employed for encryption and decryption, and further defined by their application and use. Each approach has its own advantages and disadvantages. By combining the two (symmetric and asymmetric), the advantages of both are exploited in this paper. The three fundamental types of cryptographic algorithms that will be discussed are (Fig 2):

> Symmetric (conventional, or Secret Key) cryptography
> Asymmetric (public key) cryptography
> Hash Functions

A) Secret key (symmetric) cryptography. SKC uses a single key for both encryption and decryption.

B) Public key (asymmetric) cryptography. PKC uses two keys, one for encryption and the other for decryption.



C) Hash function (one-way cryptography). Hash functions have no key since the plaintext is not recoverable from the ciphertext.

FIG 1. 2: Three types of cryptography: secret-key, public key, and hash function.

1.2.1 Symmetric Cryptography

Symmetric cryptography techniques are characterized by the fact that the same key that is used to encrypt the plain image is also used to decrypt the cipher text (noisy image). This key is referred to as a 'private key', 'secret key' or 'shared secret'. The secrecy of a private key must be maintained in order to protect against unauthorized disclosure.

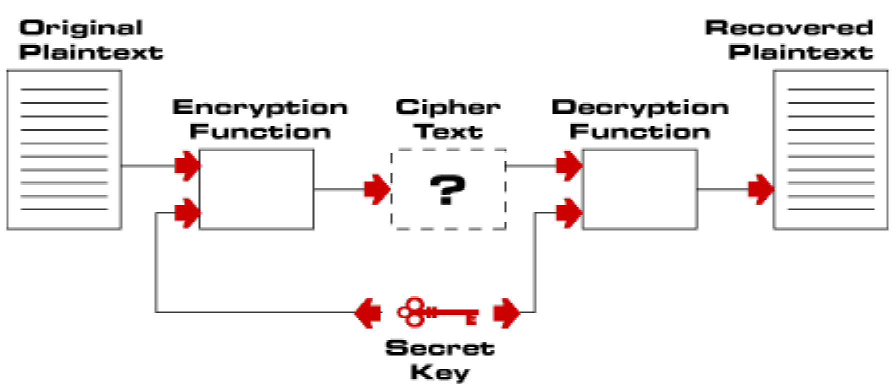


Fig 1.3. **Symmetric Cryptographic Model**

Using symmetric cryptography, it is safe to send encrypted messages without fear of interception (because an interceptor is unlikely to be able to decipher the message); however, there always remains the difficult problem of how to securely transfer the secret key to the recipients of a message so that they can decrypt the message.

The advantage of symmetric encryption is that it is fast and efficient. However, the problem is that of key exchange-the mechanism for safely ensuring that both parties, the sender and the receiver, have the secret key. The difficulty can be resolved by combining symmetric and asymmetric cryptography.

1.2.2 Asymmetric Cryptography

Asymmetric cryptography techniques are characterized by the fact that two different (but mathematically related) keys are used to perform the cryptographic operations. The asymmetric 'key pair' consists of a private key and a public key.

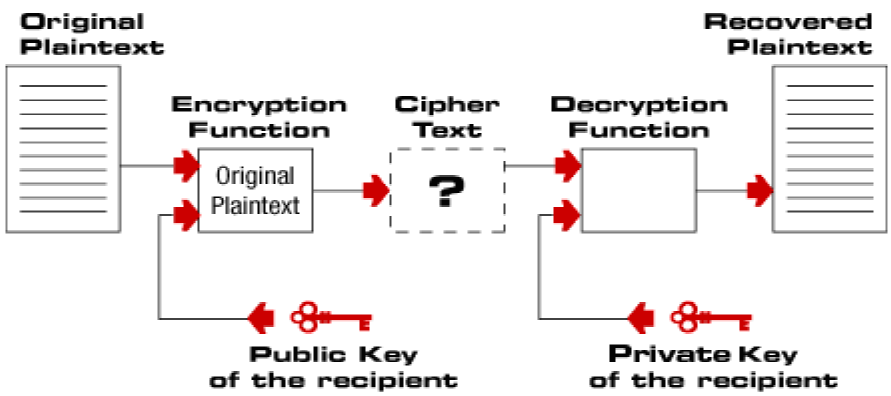


Fig 1.4 **Asymmetric Cryptography Model**

In an asymmetric scheme, each cryptographic entity (e.g. a person or a computer) has a private/public key pair. The individual user or process that owns the key is the only one that has access to the private key. The secrecy of this private key is of paramount importance for maintaining the integrity of the security services. The public key is not secret and in fact may be made available to the public at large, typically from a shared file or from a directory service.

The principal advantage of asymmetric cryptography is that key distribution is easier. The main problem with asymmetric cryptography is that the processing requires intense use of the Central Processing Unit (CPU) and this can cause potential performance problems when many simultaneous sessions are required. It can be slow and inefficient.

1.2.3 Hash Functions

It uses a mathematical transformation to irreversibly "encrypt" information. It condenses the input image irreversibly. Rather than for encryption, it is used to produce a message digest that verifies authentication.

1.3 Security Attack

To understand the types of threats to security, we need to have a definition of security requirements. Communication security addresses three requirements:

> **Confidentiality:** requires that data only be accessible for reading by authorized parties. This type of access includes printing, displaying, and other forms of disclosure, including simply revealing the existence of an object.
> **Integrity:** requires that data can be modified only by authorized parties. Modification includes writing, changing, changing status, deleting and creating.
> **Availability:** requires that data are available to authorized parties.

A very useful categorization of attacks on communication security is in terms of passive and active attacks.

1.3.1. Passive Attack

Passive attacks are in the nature of eavesdropping on, or monitoring of, transmissions. The goal of the opponent is to obtain information that is being transmitted. The two types of passive attacks are release of message contents and traffic analysis.

The **release of message contents** is easily understood. A telephone conversation, an electronic mail message, and a transferred file may contain sensitive confidential information. We would like to prevent the opponent from learning the contents of these transmissions.

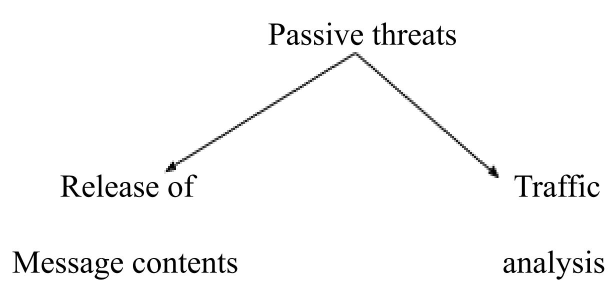


Fig 1.5 passive attack

The second passive attack, **traffic analysis**, is more subtle. Suppose that we had a way of masking the contents of messages or other information traffic so that opponents, even if they captured the message, could not extract the information from the message. The common technique for masking contents is encryption. If we had encryption protection in place, an opponent might still be able to observe the pattern of these messages. The opponent could determine the location and identity of communicating hosts and could observe the frequency and length of a message being exchanged. This information might be useful in guessing the nature of communication that was taking place.

Generally, passive attacks are very difficult to detect because they don't involve any alteration of the data. However it is feasible to prevent the success of these attacks. Thus, the emphasis in dealing with passive attacks is on prevention rather than detection.

1.3.2. **Active Attacks**

Active attacks involve some modification of the data stream or the creation of a false stream and can be subdivided into four categories: masquerade, replay, modification of messages, and denial of service.

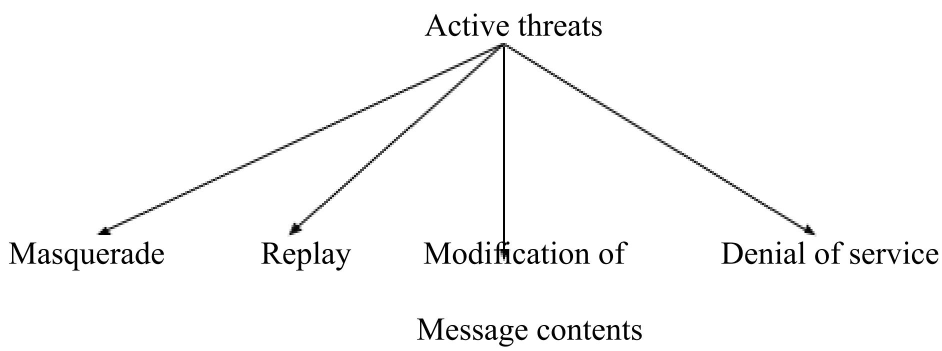


Fig 1.6 Active attack

A **masquerade** takes place when one entity pretends to be a different entity. A masquerade attack usually includes one of the other forms of the active attack. For example authentication sequences can be captured and replayed after a valid authentication sequence has taken place, thus enabling an authorized entity with few privileges to obtain extra privileges by impersonating an entity that has those privileges.

Replay involves the passive capture of a data unit and its subsequent retransmission to produce an unauthorized effect.

Modification of messages: Simply means that some portion of a legitimate message is altered or those messages are delayed or reordered, to produce unauthorized effect.

A **denial of service** attack prevents or inhibits the normal use or management of communications facilities. This attack may have a specific target; for example, an entity may suppress all messages directed to a particular destination (e.g. the security auditing service). Another form of service denial is the disruption of an entire network, either by disabling the network or by overloading it with messages to degrade performance.

Active attacks present the opposite characteristics of passive attacks. Whereas passive attacks are difficult to detect, measures are available to prevent their success. On the other hand, it is quite difficult to absolutely prevent active attacks, because to do so would require physical protection of all communications facilities and paths at all times. Instead, the goal is to detect them and to recover from any disruption or delays caused by them. Because the detection has a deterrent effect, it may also contribute to prevention.

1.3.3 Schemes of attacking conventional encryption

There are two general approaches to attacking a conventional encryption scheme. The first attack is known as **cryptanalysis,** which is the art of breaking the cipher text (or the encrypted plain text*). Cryptanalytic attack relies on the nature of the algorithm plus perhaps some knowledge of the general characteristics of the plain text or even some sample plaintext-ciphertext pairs. This type of attack exploits the characteristics of the algorithm to attempt to deduce a specific plaintext or to deduce the key being used. If the attack succeeds in deducing the key, the effect is catastrophic: all future and past messages encrypted with that key are compromised.

The second method, known as the **brute-force** attack, is to try every possible key on a piece of ciphertext until an intelligible translation into plaintext is obtained. On the average, half of all possible keys must be tried to achieve success.

Chapter-2

Conventional Encryption and Decryption

2.1 Introduction

Conventional encryption is a scheme where an intelligible text (plaintext in crypto terms) is made unintelligible (ciphertext in crypto terms) using a secure key. Block and stream ciphers and public key systems do this work. The security of the ciphers resides in the key length and decryption process is difficult without proper knowledge of the key. A conventional encryption scheme has five ingredients, namely,

- ✔ **Plaintext**: the original message or data that is fed into the algorithm as input.
- ✔ **Encryption algorithm**: the encryption algorithm performs various substitutions and transformations on the plaintext.
- ✔ **Secret key**: the secret key is also input to the encryption algorithm. The exact substitutions and transformations performed by the algorithm depend on the key.
- ✔ **Ciphertext**: this the scrambled message produced as output. It depends on the plaintext and the secret key. For a given message, two different keys will produce two different ciphertexts.
- ✔ **Decryption algorithm**: this is essentially the encryption algorithm run in reverse. It takes the ciphertext and the secret key and produces the original plaintext.

Besides, there are two requirements for secure use of conventional encryption:

1. We need a strong encryption algorithm. At the minimum, we would like the algorithm to be such that an opponent who knows the algorithm and has access to one or more ciphertexts would be unable to decipher the ciphertext or figure out the key. This requirement is usually stated in a stronger form: The opponent should be unable to decrypt the ciphertext or discover the key even if he or she is in possession of a number of ciphertexts together with the plain text that produced each cipher text.

2. Sender and receiver must have obtained copies of the secret key in a secure fashion and must keep the key secure. If someone can discover the key and knows the algorithm, all communication using this key is readable.

2.2 DES Encryption Algorithm

The DES (Data Encryption Standard) algorithm is the most widely used encryption algorithm in the world. For many years, and among many people, "secret code making" and DES have been synonymous. And despite the recent coup by the Electronic Frontier

Foundation in creating a $220,000 machine to crack DES-encrypted messages, DES will live on in government and banking for years to come through a life- extending version called "triple-DES."

How does DES work? This article explains the various steps involved in DES-encryption, illustrating each step by means of a simple example. Since the creation of DES, many other algorithms (recipes for changing data) have emerged which are based on design principles similar to DES. Once you understand the basic transformations that take place in DES, you will find it easy to follow the steps involved in these more recent algorithms.

But first a bit of history of how DES came about is appropriate, as well as a look toward the future. On May 15, 1973, during the reign of Richard Nixon, the National Bureau of Standards (NBS) published a notice in the Federal Register soliciting proposals for cryptographic algorithms to protect data during transmission and storage. The notice explained why encryption was an important issue.

Over the last decade, there has been an accelerating increase in the accumulation and communication of digital data by government, industry and by other organizations in the private sector. The contents of these communicated and stored data often have very significant value and/or sensitivity. It is now common to find data transmissions which constitute funds transfers of several million dollars, purchase or sale of securities, warrants for arrests or arrest and conviction records being communicated between law enforcement agencies, airline reservations and ticketing representing investment and value both to the airline and passengers, and health and patient care records transmitted among physicians and treatment centers.

The increasing volume, value and confidentiality of these records regularly transmitted and stored by commercial and government agencies has led to heightened recognition and concern over their exposures to unauthorized access and use. This misuse can be in the form of theft or defalcations of data records representing money, malicious modification of business inventories or the interception and misuse of confidential information about people. The need for protection is then apparent and urgent.

It is recognized that encryption (otherwise known as scrambling, enciphering or privacy transformation) represents the only means of protecting such data during transmission and a useful means of protecting the content of data stored on various media, providing encryption of adequate strength can be devised and validated and is inherently integrable into system architecture. The National Bureau of Standards solicits proposed techniques and algorithms for computer data encryption. The Bureau also solicits recommended techniques for implementing the cryptographic function: for generating, evaluating, and protecting cryptographic keys; for maintaining files encoded under expiring keys; for making partial updates to encrypted files; and mixed clear and encrypted data to permit labeling, polling, routing, etc. The Bureau in its role for establishing standards and aiding government and industry in assessing technology, will arrange for the evaluation of protection methods in order to prepare guidelines.

NBS waited for the responses to come in. It received none until August 6, 1974, three days before Nixon's resignation, when IBM submitted a candidate that it had developed internally under the name LUCIFER. After evaluating the algorithm with the help of the

National Security Agency (NSA), the NBS adopted a modification of the LUCIFER algorithm as the new Data Encryption Standard (DES) on July 15, 1977.

DES was quickly adopted for non-digital media, such as voice-grade public telephone lines. Within a couple of years, for example, International Flavors and Fragrances was using DES to protect its valuable formulas transmitted over the phone.

Meanwhile, the banking industry, which is the largest user of encryption outside the government, adopted DES as a wholesale banking standard. Standards for the wholesale banking industry are set by the American National Standards Institute (ANSI). ANSI X3.92, adopted in 1980, specified the use of the DES algorithm.

2.2.1 Some Preliminary Examples of DES

DES works on bits, or binary numbers--the 0s and 1s common to digital computers. Each group of four bits makes up a hexadecimal, or base 16, number. Binary "0001" is equal to the hexadecimal number "1", binary "1000" is equal to the hexadecimal number "8", "1001" is equal to the hexadecimal number "9", "1010" is equal to the hexadecimal number "A", and "1111" is equal to the hexadecimal number "F".

DES works by encrypting groups of 64 message bits, which is the same as 16 hexadecimal numbers. To do the encryption, DES uses "keys" which are also *apparently* 16 hexadecimal numbers long, or *apparently* 64 bits long. However, every 8th key bit is ignored in the DES algorithm, so that the effective key size is 56 bits. But, in any case, 64 bits (16 hexadecimal digits) is the round number upon which DES is organized.

For example, if we take the plaintext message "8787878787878787", and encrypt it with the DES key "0E329232EA6D0D73", we end up with the ciphertext "0000000000000000". If the ciphertext is decrypted with the same secret DES key "0E329232EA6D0D73", the result is the original plaintext "8787878787878787".

This example is neat and orderly because our plaintext was exactly 64 bits long. The same would be true if the plaintext happened to be a multiple of 64 bits. But most messages will not fall into this category. They will not be an exact multiple of 64 bits (that is, an exact multiple of 16 hexadecimal numbers).

For example, take the message "Your lips are smoother than Vaseline". This plaintext message is 38 bytes (76 hexadecimal digits) long. So this message must be padded with some extra bytes at the tail end for the encryption. Once the encrypted message has been decrypted, these extra bytes are thrown away. There are, of course, different padding schemes--different ways to add extra bytes. Here we will just add 0s at the end, so that the total message is a multiple of 8 bytes (or 16 hexadecimal digits, or 64 bits).

The plaintext message "Your lips are smoother than Vaseline" is, in hexadecimal,

"596F7572206C6970732061726520736D6F6F7468657220746861E2076617365
6C696E650D0A".

(Note here that the first 72 hexadecimal digits represent the English message, while "0D" is hexadecimal for Carriage Return, and "0A" is hexadecimal for Line Feed, showing that the message file has terminated.) We then pad this message with some 0s on the end, to get a total of 80 hexadecimal digits:

"596F7572206C6970732061726520736D6F6F7468657220746861E2076617365 6C696E650D0A0000".

If we then encrypt this plaintext message 64 bits (16 hexadecimal digits) at a time, using the same DES key "0E329232EA6D0D73" as before, we get the cipher text:

"C0999FDDE378D7ED727DA00BCA5A84EE47F269A4D6438190DD52F78F5358499 828AC9B453E0E653". This is the secret code that can be transmitted or stored. Decrypting the cipher text restores the original message "Your lips are smoother than Vaseline".

2.2.2 DES working principles

DES is a ***block cipher***--meaning it operates on plaintext blocks of a given size (64-bits) and returns ciphertext blocks of the same size. Thus DES results in a ***permutation*** among the 2^64 (read this as: "2 to the 64th power") possible arrangements of 64 bits, each of which may be either 0 or 1. Each block of 64 bits is divided into two blocks of 32 bits each, a left half block **L** and a right half **R**. (This division is only used in certain operations.) Encryption of a block of the message takes place in 16 stages or rounds. From the input key, sixteen 48 bit keys are generated, one for each round. In each round, eight so-called S-boxes are used. These S-boxes are fixed in the specification of the standard. Using the S-boxes, groups of six bits are mapped to groups of four bits. The contents of these S-boxes have been determined by the U.S. National Security Agency (NSA). The S-boxes appear to be randomly filled, but this is not the case. Recently it has been discovered that these S-boxes, determined in the 1970s, are resistant against an attack called differential cryptanalysis which was first known in the 1990s.The block of the message is divided into two halves. The right half is expanded from 32 to 48 bits using another fixed table. The result is combined with the subkey for that round using the XOR operation. Using the S-boxes the 48 resulting bits are then transformed again to 32 bits, which are subsequently permutated again using yet another fixed table. This by now thoroughly shuffled right half is now combined with the left half using the XOR operation. In the next round, this combination is used as the new left half. Figure 2.1 should hopefully make this process a bit more clear. In the figure, the left and right halves are denoted as L0 and R0, and in subsequent rounds as L1, R1, L2, R2 and so on. The function f is responsible for all the mappings described above.

To make the working principles of the algorithm more vivid, a simple example is illustrated as follows**:** Let **M** be the plain text message **M** = 0123456789ABCDEF, where **M** is in hexadecimal (base 16) format. Rewriting **M** in binary format, we get the 64-bit block of text:

M=0000000100100011010001010110011110001001101010111100110111101111

L=00000001001000110100010101100111
R = 10001001101010111100110111101111

The first bit of **M** is "0". The last bit is "1". We read from left to right.

DES operates on the 64-bit blocks using *key* sizes of 56- bits. The keys are actually stored as being 64 bits long, but every 8th bit in the key is not used (i.e. bits numbered 8, 16, 24, 32, 40, 48, 56, and 64). However, we will nevertheless number the bits from 1 to 64, going left to right, in the following calculations. But, as you will see, the eight bits just mentioned get eliminated when we create subkeys.

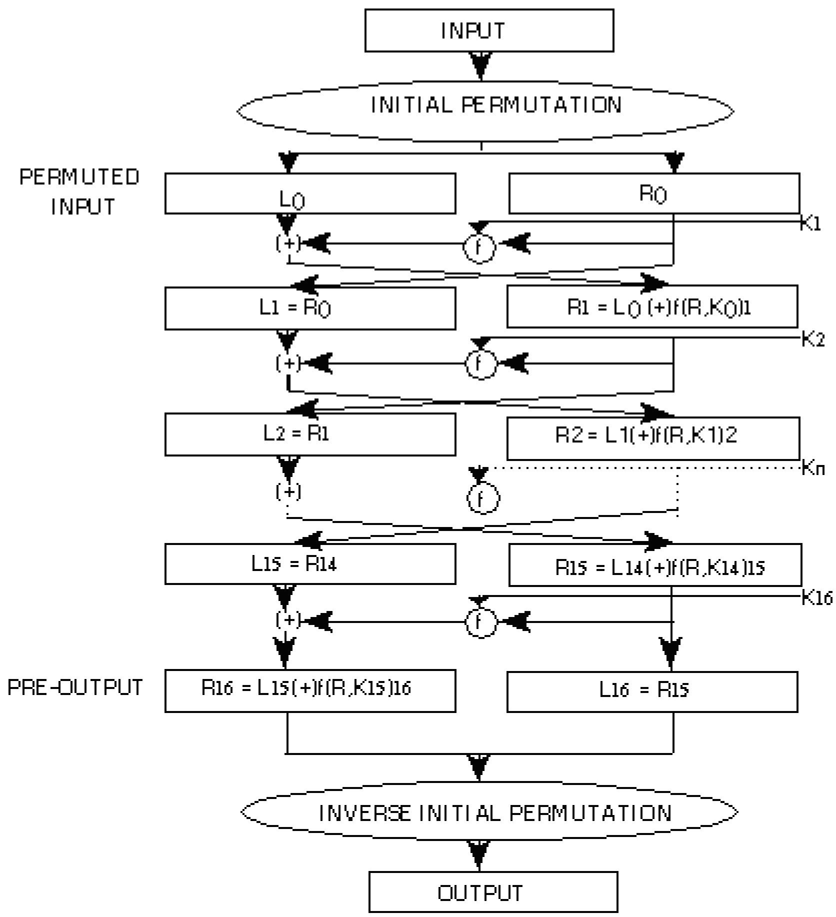


Fig 2.1 Encryption and decryption

Example: Let **K** be the hexadecimal key **K** = 133457799BBCDFF1. This gives us as the binary key (setting 1 = 0001, 3 = 0011, etc., and grouping together every eight bits, of which the last one in each group will be unused):

K = 0001001100110100010101110111100110011011101111001101111111110001

Each of the steps involved in the DES algorithm are illustrated in detail as follows:

Step 1: Creating 16 subkeys, each of which is 48-bits long (key scheduling).

The key passes through certain processes as depicted in figure 2.2 before it is used for the process of encryption.

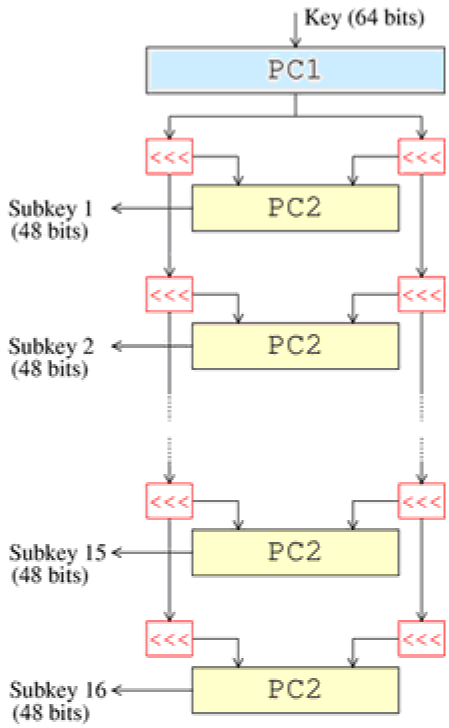


Fig 2.2 — The key-schedule of DES

The 64-bit key is permuted according to the following table, **PC-1**. Since the first entry in the table is "57", this means that the 57th bit of the original key **K** becomes the first bit of the permuted key **K**+. The 49th bit of the original key becomes the second bit of the permuted key. The 4th bit of the original key is the last bit of the permuted key. Note only 56 bits of the original key appear in the permuted key.

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

Example: From the original 64-bit key

K = 0001001100110100010101110111100110011011101111001101111111110001

we get the 56-bit permutation

K+ = 11110000110011001010101011110101010101100110011110001111

Next, split this key into left and right halves, C_0 and D_0, where each half has 28 bits.

Example: From the permuted key **K**+, we get

$\boldsymbol{C_0}$=1111000011001100101010101111
$\boldsymbol{D_0}$ = 0101010101100110011110001111

With $\boldsymbol{C_0}$ and $\boldsymbol{D_0}$ defined, we now create sixteen blocks $\boldsymbol{C_n}$ and $\boldsymbol{D_n}$, 1<=$\boldsymbol{n}$<=16. Each pair of blocks $\boldsymbol{C_n}$ and $\boldsymbol{D_n}$ is formed from the previous pair $\boldsymbol{C_{n-1}}$ and $\boldsymbol{D_{n-1}}$, respectively, for $\boldsymbol{n}$ = 1, 2, ..., 16, using the following schedule of "left shifts" of the previous block. To do a left shift, move each bit one place to the left, except for the first bit, which is cycled to the end of the block.

Iteration Number	Number of Left Shifts
1	1
2	1
3	2
4	2
5	2
6	2
7	2
8	2
9	1
10	2
11	2
12	2
13	2
14	2
15	2
16	1

This means, for example, $\boldsymbol{C_3}$ and $\boldsymbol{D_3}$ are obtained from $\boldsymbol{C_2}$ and $\boldsymbol{D_2}$, respectively, by two left shifts, and $\boldsymbol{C_{16}}$ and $\boldsymbol{D_{16}}$ are obtained from $\boldsymbol{C_{15}}$ and $\boldsymbol{D_{15}}$, respectively, by one left shift. In all cases, a single left shift means a rotation of the bits one place to the left, so that after one left shift the bits in the 28 positions are the bits that were previously in positions 2, 3,..., 28, 1.

Example: From original pair $\boldsymbol{C_0}$ and $\boldsymbol{D_0}$ we obtain:

$\boldsymbol{C_0}$=1111000011001100101010101111
$\boldsymbol{D_0}$ = 0101010101100110011110001111

$\boldsymbol{C_1}$ = 1110000110011001010101011111
$\boldsymbol{D_1}$ = 1010101011001100111100011110

$\boldsymbol{C_2}$ = 1100001100110010101010111111
$\boldsymbol{D_2}$ = 0101010110011001111000111101

$\boldsymbol{C_3}$ = 0000110011001010101011111111
$\boldsymbol{D_3}$ = 0101011001100111100011110101

$\boldsymbol{C_4}$ = 0011001100101010101111111100
$\boldsymbol{D_4}$ = 0101100110011110001111010101

$\boldsymbol{C_5}$ = 1100110010101010111111110000
$\boldsymbol{D_5}$ = 0110011001111000111101010101

C_6 = 0011001010101011111111000011
D_6 = 1001100111100011110101010101

C_7 = 1100101010101111111100001100
D_7 = 0110011110001111010101010110

C_8 = 0010101010111111110000110011
D_8 = 1001111000111101010101011001

C_9 = 0101010101111111100001100110
D_9 = 0011110001111010101010110011

C_{10} = 0101010111111110000110011001
D_{10} = 1111000111101010101011001100

C_{11} = 0101011111111000011001100101
D_{11} = 1100011110101010101100110011

C_{12} = 0101111111100001100110010101
D_{12} = 0001111010101010110011001111

C_{13} = 0111111110000110011001010101
D_{13} = 0111101010101011001100111100

C_{14} = 1111111000011001100101010101
D_{14} = 1110101010101100110011110001

C_{15} = 1111100001100110010101010111
D_{15} = 1010101010110011001111000111

C_{16} = 1111000011001100101010101111
D_{16} = 0101010101100110011110001111

We now form the keys K_n, for 1<=n<=16, by applying the following permutation table to each of the concatenated pairs C_nD_n. Each pair has 56 bits, but **PC-2** only uses 48 of these.

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

Therefore, the first bit of K_n is the 14th bit of C_nD_n, the second bit the 17th, and so on, ending with the 48th bit of K_n being the 32nd bit of C_nD_n.

Example: For the first key we have

C_1D_1 = 11100001100110010101010111111010101011001100111100011110

Which, after we apply the permutation **PC-2**, becomes

K_1 = 000110110000001011101111111111000111000001110010

For the other keys we have

K_2 = 011110 011010 111011 011001 110110 111100 100111 100101
K_3 = 010101 011111 110010 001010 010000 101100 111110 011001
K_4 = 011100 101010 110111 010110 110110 110011 010100 011101
K_5 = 011111 001110 110000 000111 111010 110101 001110 101000
K_6 = 011000 111010 010100 111110 010100 000111 101100 101111
K_7 = 111011 001000 010010 110111 111101 100001 100010 111100
K_8 = 111101 111000 101000 111010 110000 010011 101111 111011
K_9 = 111000 001101 101111 101011 111011 011110 011110 000001
K_{10} = 101100 011111 001101 000111 101110 100100 011001 001111
K_{11} = 001000 010101 111111 010011 110111 101101 001110 000110
K_{12} = 011101 010111 000111 110101 100101 000110 011111 101001
K_{13} = 100101 111100 010111 010001 111110 101011 101001 000001
K_{14} = 010111 110100 001110 110111 111100 101110 011100 111010
K_{15} = 101111 111001 000110 001101 001111 010011 111100 001010
K_{16} = 110010 110011 110110 001011 000011 100001 011111 110101

So much for the subkeys. Now we look at the message itself.

Step 2: Encoding each 64-bit block of data.

There is an *initial permutation* **IP** of the 64 bits of the message data **M**. This rearranges the bits according to the following table, where the entries in the table show the new arrangement of the bits from their initial order. The 58th bit of **M** becomes the first bit of **IP**. The 50th bit of **M** becomes the second bit of **IP**. The 7th bit of **M** is the last bit of **IP**.

IP

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
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

Example: Applying the initial permutation to the block of text **M**, given previously, we get

M=0000000100100011010001010110011110001001101010111100110111101111
IP=1100110000000000110011001111111111110000101010101111000010101010

Here the 58th bit of **M** is "1", which becomes the first bit of **IP**. The 50th bit of **M** is "1", which becomes the second bit of **IP**. The 7th bit of **M** is "0", which becomes the last bit of **IP**.

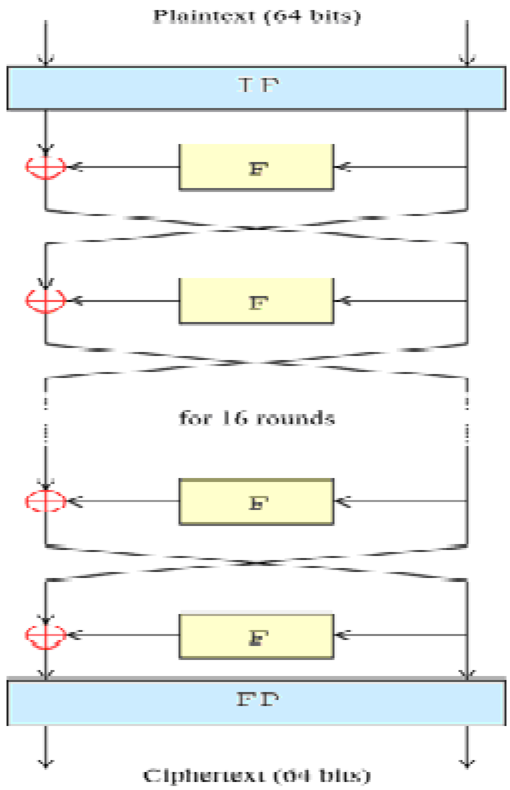


Fig 2.3 The overall Feistel structure of DES

Next divide the permuted block **IP** into a left half L_0 of 32 bits, and a right half R_0 of 32 bits.

Example: From **IP**, we get L_0 and R_0

$$L_0 = 1100\ 1100\ 0000\ 0000\ 1100\ 1100\ 1111\ 1111$$
$$R_0 = 1111\ 0000\ 1010\ 1010\ 1111\ 0000\ 1010\ 1010$$

We now proceed through 16 iterations, for 1<=n<=16, using a function f which operates on two blocks--a data block of 32 bits and a key K_n of 48 bits--to produce a block of 32 bits. **Let + denote XOR addition, (bit-by-bit addition modulo 2)**. Then for **n** going from 1 to 16 we calculate

$$L_n = R_{n-1}$$
$$R_n = L_{n-1} + f(R_{n-1}, K_n)$$

This results in a final block, for n = 16, of $L_{16}R_{16}$. That is, in each iteration, we take the right 32 bits of the previous result and make them the left 32 bits of the current step. For the right 32 bits in the current step, we XOR the left 32 bits of the previous step with the calculation f.

Example: For n = 1, we have

$$K_1 = 000110110000001011101111111111000111000001110010$$
$$L_1 = R_0 = 11110000101010101111000010101010$$
$$R_1 = L_0 + f(R_0,K_1)$$

It remains to explain how the function f works. To calculate f, we first expand each block R_{n-1} from 32 bits to 48 bits. This is done by using a selection table that repeats some of the bits in R_{n-1} . We'll call the use of this selection table the function **E**. Thus $\mathbf{E}(R_{n-1})$ has a 32 bit input block, and a 48 bit output block.

The F-function, depicted in Figure 2.4, operates on half a block (32 bits) at a time and consists of four stages:

1. *Expansion* — the 32-bit half-block is expanded to 48 bits using the *expansion permutation*, denoted *E* in the diagram, by duplicating some of the bits.
2. *Key mixing* — the result is combined with a *subkey* using an XOR operation. Sixteen 48-bit sub keys — one for each round — are derived from the main key using the key scheduling.
3. *Substitution* — after mixing in the sub key, the block is divided into eight 6-bit pieces before processing by the S-boxes, or *substitution boxes*. Each of the eight S-boxes replaces its six input bits with four output bits according to a non-linear transformation, provided in the form of a lookup table. The S-boxes provide the core of the security of DES — without them, the cipher would be linear, and trivially breakable.
4. *Permutation* — finally, the 32 outputs from the S-boxes are rearranged according to a fixed permutation, the *P-box*.

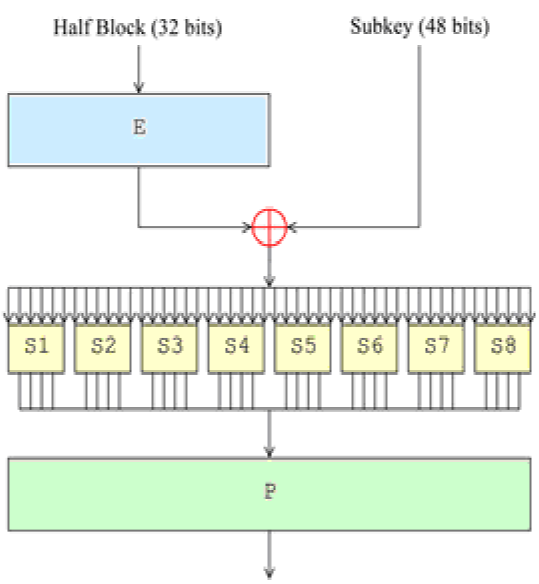


Fig 2.4 The Feistel function (F-function) of DES

Let **E** be such that the 48 bits of its output, written as 8 blocks of 6 bits each, are obtained by selecting the bits in its inputs in order according to the following table:

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

Thus the first three bits of $\mathbf{E}(\boldsymbol{R_{n-1}})$ are the bits in positions 32, 1 and 2 of $\boldsymbol{R_{n-1}}$ while the last 2 bits of $\mathbf{E}(\boldsymbol{R_{n-1}})$ are the bits in positions 32 and 1.

Example: We calculate $\mathbf{E}(\boldsymbol{R_0})$ from $\boldsymbol{R_0}$ as follows:

$\boldsymbol{R_0}$ = 11110000101010101111000010101010
$\mathbf{E}(\boldsymbol{R_0})$ = 011110100001010101010101011110100001010101010101

(Note that each block of 4 original bits has been expanded to a block of 6 output bits.)

Next in the $\boldsymbol{f}$ calculation, we XOR the output $\mathbf{E}(\boldsymbol{R_{n-1}})$ with the key $\boldsymbol{K_n}$:

$$\boldsymbol{K_n} + \mathbf{E}(\boldsymbol{R_{n-1}}).$$

Example: For $\boldsymbol{K_1}$, $\mathbf{E}(\boldsymbol{R_0})$, we have

$\boldsymbol{K_1}$ = 000110110000001011101111111111000111000001110010
$\mathbf{E}(\boldsymbol{R_0})$ = 011110100001010101010101011110100001010101010101
$\boldsymbol{K_1}$+$\mathbf{E}(\boldsymbol{R_0})$ = 011000010001011110111010100001100110010100100111.

We have not yet finished calculating the function $\boldsymbol{f}$. To this point we have expanded $\boldsymbol{R_{n-1}}$ from 32 bits to 48 bits, using the selection table, and XORed the result with the key $\boldsymbol{K_n}$. We now have 48 bits, or eight groups of six bits. We now do something strange with each group of six bits: we use them as addresses in tables called "**S boxes**". Each group of six bits will give us an address in a different **S** box. Located at that address will be a 4 bit number. This 4 bit number will replace the original 6 bits. The net result is that the eight groups of 6 bits are transformed into eight groups of 4 bits (the 4-bit outputs from the **S** boxes) for 32 bits total.

Write the previous result, which is 48 bits, in the form:

$$\boldsymbol{K_n} + \mathbf{E}(\boldsymbol{R_{n-1}}) = \boldsymbol{B_1B_2B_3B_4B_5B_6B_7B_8},$$

where each $\boldsymbol{B_i}$ is a group of six bits. We now calculate

$$\boldsymbol{S_1(B_1)S_2(B_2)S_3(B_3)S_4(B_4)S_5(B_5)S_6(B_6)S_7(B_7)S_8(B_8)}$$

where $\boldsymbol{S_i(B_i)}$ refers to the output of the $\boldsymbol{i}$-th **S** box.

To repeat, each of the functions ***S1, S2,..., S8***, takes a 6-bit block as input and yields a 4-bit block as output. The table to determine $\boldsymbol{S_1}$ is shown and explained below:

S1

Column Number

Row

No.	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

If S_1 is the function defined in this table and B is a block of 6 bits, then $S_1(B)$ is determined as follows: The first and last bits of B represent in base 2 a number in the decimal range 0 to 3 (or binary 00 to 11). Let that number be i. The middle 4 bits of B represent in base 2 a number in the decimal range 0 to 15 (binary 0000 to 1111). Let that number be j. Look up in the table the number in the i-th row and j-th column. It is a number in the range 0 to 15 and is uniquely represented by a 4 bit block. That block is the output $S_1(B)$ of S_1 for the input B. For example, for input block B = 011011 the first bit is "0" and the last bit "1" gives 01 as the row. This is row 1. The middle four bits are "1101". This is the binary equivalent of decimal 13, so the column is column number 13. In row 1, column 13 appears 5. This determines the output; 5 is binary 0101, so that the output is 0101. Hence $S_1(011011) = 0101$.

The tables defining the functions $S_1,...,S_8$ are the following:

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	9

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

Example: For the first round, we obtain as the output of the eight **S** boxes:

$\boldsymbol{K_1} + \mathbf{E}(\boldsymbol{R_0})$ = 011000 010001 011110 111010 100001 100110 010100 100111.

$\boldsymbol{S_1(B_1)S_2(B_2)S_3(B_3)S_4(B_4)S_5(B_5)S_6(B_6)S_7(B_7)S_8(B_8)}$ = 0101 1100 1000 0010 1011 0101 1001 0111

The final stage in the calculation of ***f*** is to do a permutation **P** of the **S**-box output to obtain the final value of ***f***:

$$\boldsymbol{f} = \mathbf{P}(\boldsymbol{S_1(B_1)S_2(B_2)...S_8(B_8)})$$

The permutation **P** is defined in the following table. **P** yields a 32-bit output from a 32-bit input by permuting the bits of the input block.

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

Example: From the output of the eight **S** boxes:

$\boldsymbol{S_1(B_1)S_2(B_2)S_3(B_3)S_4(B_4)S_5(B_5)S_6(B_6)S_7(B_7)S_8(B_8)}$ = 0101 1100 1000 0010 1011 0101 1001 0111

we get

$$\boldsymbol{f} = 0010\ 0011\ 0100\ 1010\ 1010\ 1001\ 1011\ 1011$$

$$\boldsymbol{R_1} = \boldsymbol{L_0} + \boldsymbol{f}(\boldsymbol{R_0}, \boldsymbol{K_1})$$

$$\begin{aligned} &= 1100\ 1100\ 0000\ 0000\ 1100\ 1100\ 1111\ 1111 \\ &+ 0010\ 0011\ 0100\ 1010\ 1010\ 1001\ 1011\ 1011 \\ &= 1110\ 1111\ 0100\ 1010\ 0110\ 0101\ 0100\ 0100 \end{aligned}$$

In the next round, we will have $L_2 = R_1$, which is the block we just calculated, and then we must calculate $R_2 = L_1 + f(R_1, K_2)$, and so on for 16 rounds. At the end of the sixteenth round we have the blocks L_{16} and R_{16}. We then ***reverse*** the order of the two blocks into the 64-bit block

$$R_{16}L_{16}$$

and apply a final permutation **IP^{-1}** as defined by the following table:

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

That is, the output of the algorithm has bit 40 of the preoutput block as its first bit, bit 8 as its second bit, and so on, until bit 25 of the pre-output block is the last bit of the output.

Example: If we process all 16 blocks using the method defined previously, we get, on the 16th round,

L_{16} = 0100 0011 0100 0010 0011 0010 0011 0100
R_{16} = 0000 1010 0100 1100 1101 1001 1001 0101

We reverse the order of these two blocks and apply the final permutation to

$R_{16}L_{16}$ = 0000101001001100110110011001010101000011010000100011001000110100

IP^{-1} = 1000010111101000000100110101010000001111000010101011010000000101

which in hexadecimal format is

85E813540F0AB405.

This is the encrypted form of **M** = 0123456789ABCDEF: namely,

C = 85E813540F0AB405

2.3 Decryption

The process of decryption with DES is essentially the same as the encryption process. The rule is as follows: Use the cipher text as input to the DES algorithm, but use the keys K_i in reverse order. That is, use K_{16} on the first iteration, K_{15} on the second iteration, and so on until K_1 is used on the sixteenth and last iteration.

Chapter-3
Public key Encryption and RSA algorithm

3.1 Public key encryption

Of equal importance to conventional encryption is public key encryption also called symmetric key encryption? Public key encryption, first publicly proposed by Diffie and Hellman in 1976, is the first truly revolutionary advance in encryption in literally thousands of years. For one thing, public key algorithms are based on mathematical functions rather than on substitution and permutations .But more importantly, public key cryptography is asymmetric, involving the use of two separate keys as clearly shown in fig3.1, in contrast to the conventional symmetric key, which uses only one key. The use of two keys has profound consequences in the areas of confidentiality, key distribution, and authentication.

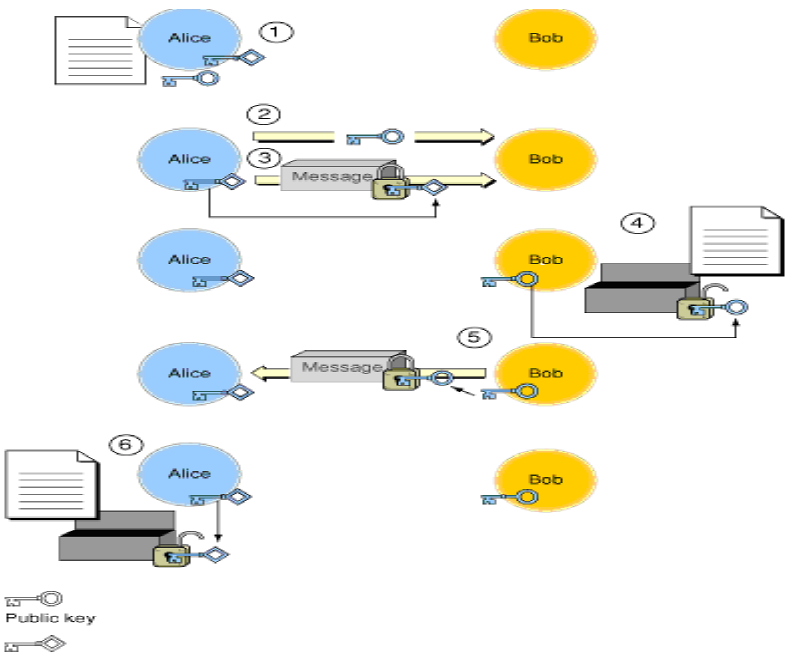


Fig 3.1 Asymmetric encryption.

Before proceeding, we should mention several common misconceptions concerning public- key encryption. One such misconception is that public-key encryption is more secure from cryptanalysis than conventional encryption. In fact, the security of any encryption scheme depends on the length of the key and the computational work involved in breaking a cipher .There is nothing in principle about conventional or public –key encryption that makes one superior to another from the point of view of cryptanalysis. A second misconception is that public-key encryption is a general purpose technique that has made conventional encryption obsolete. On the contrary, because of the computational overhead of current public-key encryption schemes, there seems no foreseeable likelihood that conventional encryption will be abandoned. Finally, there is a feeling that key distribution is trivial when using public-key encryption ,compared to the rather cumbersome handshaking involved with key distribution centers for conventional and the procedure involved are no simpler nor any more efficient than those required for conventional encryption:

There are a couple of glaring problems with symmetric encryption. Firstly, it requires a separate key for each pair of users (or group of users), which is okay in the schoolyard, but becomes quite a management problem when you scale it globally. Secondly, there's no easy way to share the key securely in the first place. In the schoolyard, you can just hand it to one another, making sure no-one else sees it. On the Internet, especially when you want to share messages with many people, physically handing the key over is not feasible. Nor can you do it by email or phone, because both are themselves insecure means of delivery. So you're in a catch-22. One solution for this is asymmetric *encryption*, also known as *public key encryption*. In this form of encryption, each person has a pair of keys (see figure 3.1). One key is a public key, which can be made freely available, even advertised in a directory for all to see. The other key, which you guard carefully, is a private key. A message encoded with a particular public key can only be decoded using the corresponding private key, and vice versa. To send a message to someone using public key encryption:
Acquire the recipient's public key.

1. Create your message.
2. Encrypt it with the recipient's public key. Once it's encoded, it can only be decoded with the recipient's private key. It doesn't matter if someone intercepts the message, as they won't have the private key needed to decode it.
3. Send the encrypted message to the recipient.
4. The recipient decodes the message using their private key.

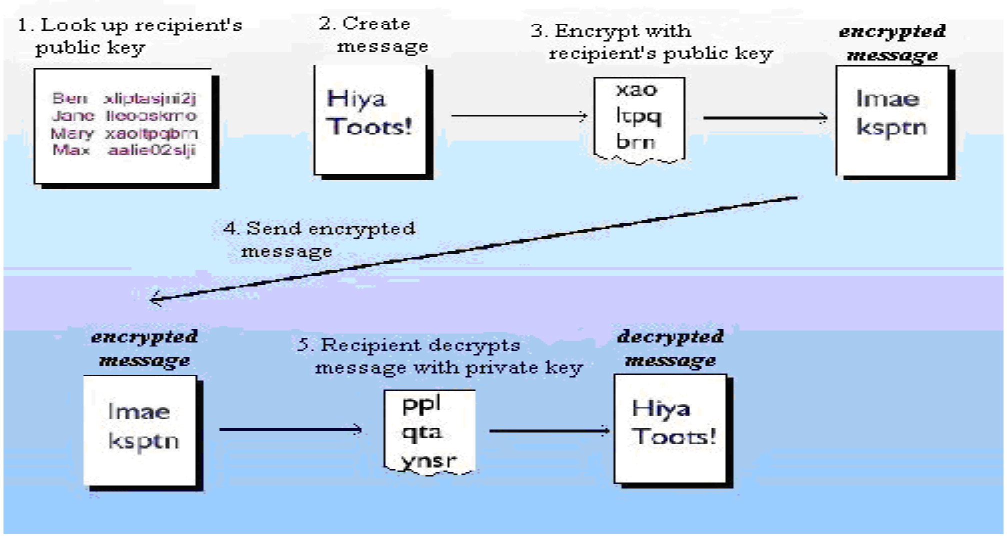


Fig3.2 sending secure messages with public key encryption

Not only can you use public key encryption to send private messages, you can also use it to let the recipient know the message really is from you, and not from some imposter. Here's how:

1. Create your message.
2. Encrypt it using your private key.
3. Send the encrypted message to the recipient.
4. The recipient looks up your *public* key and uses this to decode the message. If the message decodes correctly using your public key, they can be sure that the message was created using your private key, and thus that it originates from you.

A public–key cryptographic algorithm relies on one key for encryption and a different but related key for decryption. Further, these algorithms have the following important characteristics.

It is computationally infeasible to determine the decryption key given only knowledge of the cryptographic algorithm and the encryption key.
In addition, some algorithms, such as RSA, also exhibit the following characteristics.

Either of the two related keys can be used for encryption, with the other used for decryption.

With this approach, all participants have access to public keys, and private keys are generated locally by each participant and therefore need never be distributed .As long as a system controls its private key, its incoming communication is secure. At any time, a system can change its private key and publish the companion public key to replace its old public key.

A public key encryption scheme has six ingredients as shown below.

- **Plaintext:** This is the readable message or data that is fed into the algorithm as input.
- **Encryption algorithm:** The encryption algorithm performs various transformations on the plain text.
- **Public and private key:** This is a pair of keys that have been selected so that if one is used for encryption the other is used for decryption .The exact transformations performed by the encryption algorithm depend on the public or private key that is provided as input.
- **Cipher text :** This is the scrambled message produced out put .It depends on the plain text and the key for a given message ,two different keys will produce two different cipher texts.
- **Decryption algorithm:** This algorithm accepts the cipher text and the matching key and produces the original plain text.

3.2 RSA

3.2.1 RSA overview

RSA is a public key cryptosystem for both encryption and authentication .It was invented in 1977 by Ron **R**ivest, Adi **S**hamir and Leon **A**dleman (US patent 4,405,829).It is based on the assumption that it is easy to multiply two prime numbers, but difficult to divide the result again into the two prime numbers .It is an encryption algorithm that uses very large prime numbers to generate the public key and the private key. RSA is typically used in conjunction with a secret key cryptosystem such as DES .DES would be used to encrypt the message as a whole and then use RSA to encrypt the secret key. Thus, RSA provides a digital envelope for the message. RSA is in wide use today ,it is possibly the most commonly used public key algorithm used .Because of this it has undergone a lot of public security and there is much empirical evidence of its security .It can be used for both encryption and signing.

Although it would be possible to factor out the public key to get the private key (2 prime factors must be found out), the numbers are so large as to make it very practical to do so. The encryption algorithm itself is very slow, which makes it impractical to use RSA to encrypt large data sets .In PGP (and most other RSA-based encryption program), asymmetrical key is encrypted using the public key, and then the remainder of the data is encrypted with a faster algorithm using the symmetrical key .The symmetrical key itself is randomly generated, so that the only way to get it would be by using the private key to decrypt the RSA-encrypted symmetrical key.

The RSA system involves several calculations modulo a certain number .Calculating modulo some number means that the outcome of a calculation can never be larger than this number .A common example is second and minute on the clock. If it is now 6:58 and we add five minutes, the outcome is 7:03 and not 6:63.Time is thus calculated modulo 60. Using modulo calculation means that the mathematical computations can be implemented very efficiently.

3.2.2 RSA Algorithm

The RSA algorithm can be used for both public key encryption and digital signatures. Its security is based on the difficulty of factoring large integers.

i. RSA Key generation algorithm

1. Generate two large random primes, p and q, of approximately equal size such that their product n=p*q is of the required bit length , e.g 1024 bits.
2. Compute n=pq and (Φ) phi= (p-1) (q-1).
3. Choose an integer e, 1<e<phi, such that gcd (e, phi) =1. (See appendix...)

4. Compute the secret exponent d using extended Euclidean algorithm, 1<d<phi, such that ed=1(mod phi). (See appendix...)
5. The public key is (n, e) and the private key is (n, d).The values of p, q, and phi should also be kept secret.

- N is known as the *public exponent* or *encryption exponent*.
- E is known as the *secret exponent* or *decryption exponent*.

ii. RSA public –key encryption

Sender A does the following:-

1. Obtain the recipient B' public key (n, e).
2. Represent the plaintext message as a positive integer m.
3. Computes the cipher text $c=m^e modn$.
4. Send the cipher text c to B.

iii. RSA public-key decryption

Recipient B does the following:-

1. Uses his private key (n, d) to compute $m=c^{d} \bmod n$.
2. Extracts the plaintext from the integer representative m.

3.2.3 Key Management

With conventional encryption, a fundamental requirement for two parties to communicate securely is they share a secret key. Suppose Alice wants to create a messaging application that will enable him to exchange email securely with anyone who has access to the Internet or to some other network that the two of them share. Suppose Alice wants to do this using only conventional encryption. With conventional encryption ,Alice and her correspondent , say, Bob must come up with a way to share a unique secret key that no one else knows .How are they going to do that ? If Bob is in the next room from Alice, Alice could generate a key and write it down on a piece of paper or store it on a diskette and hand it to Bob .But if Bob is on the other side of the continent or the world , what can Alice do ? She could encrypt this key using conventional encryption and email it to Bob, But this means that Alice and Bob must share a secret key to encrypt this new secret key. Furthermore, Alice and every one else who uses this new email key package faces the same problem with every potential correspondent: Each pair of correspondents must share a unique secret key.

How to distribute secret keys securely is the most difficult problem for conventional encryption. This problem is wiped away with public- key encryption by the simple fact that the private key is never distributed .If Alice wants to correspond with Bob and other people; she generates a single pair of keys, one private and one public. She keeps the private key secure and broadcasts the public key to all and sundry .If Bob does the same, then Alice has Bob' public key, Bob has Alice's public key, and they can now communicate securely. When Alice wishes to communicate with Bob, Alice can do the following:

1. Prepare a message.
2. Encrypt that message using conventional encryption with a one –time conventional session key.
3. Encrypt the session key using public key encryption with Bob's public key.
4. Attach the encrypted session key to the message and send it to Bob.

Only Bob is capable of decrypting the session key and therefore of recovering the original message.

It is only fair to point out, however, that we have replaced one problem with another. Bob's private key is secure because he need never reveal it; however, Alice must be sure that the public key with Bob's name written all over it is in fact Bob's public key .Someone else could have broadcast a public key and said it was Bob's. The common way to overcome this problem is ingenious: Use public key encryption to authenticate the public key. This assumes the existence of some trusted signing authority or individual and works as follows:

1. A public key is generated by Bob and submitted to agency X for certification
2. X determines by some procedure, such as a face to face meeting, that this is authentically Bob's public key.
3. X appends a timestamp to the public key , generates the hash code of the result , and encrypts that result with its private key ,forming the signature .result , and encrypts that result with its private key , forming the signature .
4. The signature is attached to the public key.

Any one equipped with a copy of X's public key can now verify that Bob's public key is authentic.

3.2.4 Example of RSA encryption

As it has been described above, the security of the RSA algorithm comes from the computational difficulty of factoring very large numbers .Hence to be more secure large numbers must be used for p and q – 100 decimal digits at the very least.

Now let's go through a simple worked example

1) Generate two large prime numbers, p and q

To make the example easy to follow we are going to use small numbers, but this is not secure. To find random primes, we start at a random number and go up ascending odd numbers until we find a prime (see also the appendix for generating prime numbers). Let's have:

$p = 7$
$q = 19$

*2) Let n = p*q*

n = 7 * 19
= 133

3) Let Φ = (p - 1) (q - 1)

Φ = (7 - 1) (19 - 1)
= 6 * 18
= 108

4) Choose a small number, e co prime to Φ

e coprime to ***Φ***, means that the largest number that can exactly divide both e and ***Φ*** (their greatest common divisor, or gcd) is 1. Euclid's algorithm (see appendix) is used to find the gcd of two numbers, but the details are omitted here.

e = 2 => gcd(e, 108) = 2 (no)
e = 3 => gcd(e, 108) = 3 (no)
e = 4 => gcd(e, 108) = 4 (no)
e = 5 => gcd(e, 108) = 1 (yes!)

5) Find d, such that de % Φ = 1

This is equivalent to finding d which satisfies $de = 1 + n\,\Phi$ where n is any integer. We can rewrite this as $d = (1 + n\,\Phi) / e$. Now we work through values of n until an integer solution for e is found:

n = 0 => d = 1 / 5 (no)
n = 1 => d = 109 / 5 (no)
n = 2 => d = 217 / 5 (no)
n = 3 => d = 325 / 5
= 65 (yes!)

To do this with big numbers, a more sophisticated algorithm called extended Euclid (see appendix)must be used. Hence the generated pair of keys is

Public Key	**Secret Key**
n = 133 e = 5	n = 133 d = 65

A. Encryption

The message must be a number less than the smaller of p and q. However, at this point we don't know p or q, so in practice a lower bound on p and q must be published. This can be somewhat below their true value and so isn't a major security concern. For this example, let's use the message "6".

$$
\begin{aligned}
C &= P^e \% n \\
&= 6^5 \% 133 \\
&= 7776 \% 133 \\
&= 62
\end{aligned}
$$

B. Decryption

This works very much like encryption, but involves a larger exponential, which is broken down into several steps.

$$
\begin{aligned}
P &= C^d \% n \\
&= 62^{65} \% 133 \\
&= 62 * 62^{64} \% 133 \\
&= 62 * (62^2)^{32} \% 133 \\
&= 62 * 3844^{32} \% 133 \\
&= 62 * (3844 \% 133)^{32} \% 133 \\
&= 62 * 120^{32} \% 133
\end{aligned}
$$

We now repeat the sequence of operations that reduced 62^{65} to 120^{32} to reduce the exponent down to 1.

$$
\begin{aligned}
&= 62 * 36^{16} \% 133 \\
&= 62 * 99^8 \% 133 \\
&= 62 * 92^4 \% 133 \\
&= 62 * 85^2 \% 133 \\
&= 62 * 43 \% 133 \\
&= 2666 \% 133 \\
&= 6
\end{aligned}
$$

And that matches the plaintext we put in at the beginning, so the algorithm worked!

Chapter-4
Message Authentication and Hash Function

Encryption protects against passive attack (eavesdropping). A different requirement is to protect against active attacks (falsification of data and transactions). Protection against such attacks is known as message authentication
A message file, document or other collection of data is said to be authentic when it is genuine and came from its alleged source. Message authentication is a procedure that allows communicating parties to verify that received messages are authentic. The two important aspects are to verify that the contents of the message have not been altered and that the source is authentic

4.1 .Digital signatures

Digital signatures are an analog of handwritten signatures. *Digital signature* can reference the same definition used for electronic signatures `electronic signature' means an electronic sound, symbol, or process, attached to or logically associated with a contract or other record and executed or adopted by a person with the intent to sign the record .Handwritten signatures are based on the physically idiosyncratic way of signing one's name. But they can be easily forged. A digital signature is a mathematically precise way of attaching one's identity to a message. It is much harder to forge than a handwritten signature, and the signed message cannot be altered without also invalidating the signature.

Digital signatures rely on public key cryptography. Public key cryptography uses two "keys". Take an ordinary plain-text message and apply one of the keys to it in an encryption process, and you end up with a scrambled or "encrypted" (or, in the current context, "signed") message. Apply the other key to the scrambled message in a decryption process, and you end up with the original plain-text message.

One of these two keys is *'public."* Everyone knows what this key is--or can look it up, much like a telephone number. The other key is *'private."* Only you know your private key. Signing (encrypting) something with your private key puts your personal stamp on it. No one else can do this, because they don't know your private key.

A digital signature is (most often) a message digest encrypted with someone's private key to certify the contents. (A *message digest (cryptographic hash code)* is nothing more than a number -a special number that is effectively a hash code produced by a function that is very difficult to reverse.). This process of encryption is called signing. Digital signatures can perform two different functions, both very important to the security of your system:

- *Integrity:* A digital signature indicates whether a file or a message has been modified.

- *Authentication:* A digital signature makes possible mathematically verifying the name of the person who signed the message.

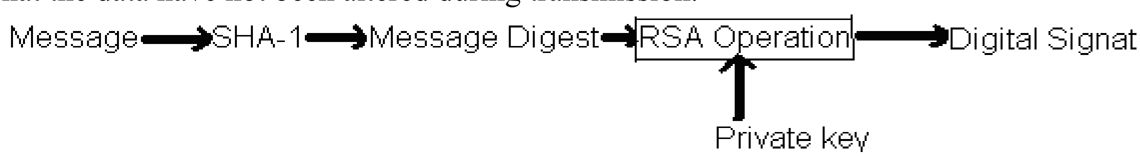
- A third function that is quite valuable in some contexts is called *non-repudiation.* Non- repudiation means that after you have signed and sent a message, you cannot later claim that you did not sign the original message. You cannot repudiate your signature, because the message was signed with your private key (which, presumably, no one else has).

The most common digital signature technology currently in use is the Digital Signature Standard (DSS), which was accepted by the US Government as the official technology for federal electronic communications. The DSS specifies the use of the Digital Signature Algorithm (DSA) and the Secure Hash Algorithm (SHA-l) to produce digital signatures.

4.2. Calculation of Digital Signatures

1- Hash

A digital signature is created by fingerprinting electronic data using a mathematical formula, known as a hash algorithm. The Secure Hash Algorithm, used in the DSS, is one of several different algorithms available. The algorithm produces a message digest or "hash" which is derived from, and unique to, the data. Consequently, the digest is unique to a particular document and cannot be duplicated. When the digest is attached to the document, the document has been "signed". Reciepients can later use the digest to verify that the data have not been altered during transmission.



.

Fig.4.1 Signature Generation

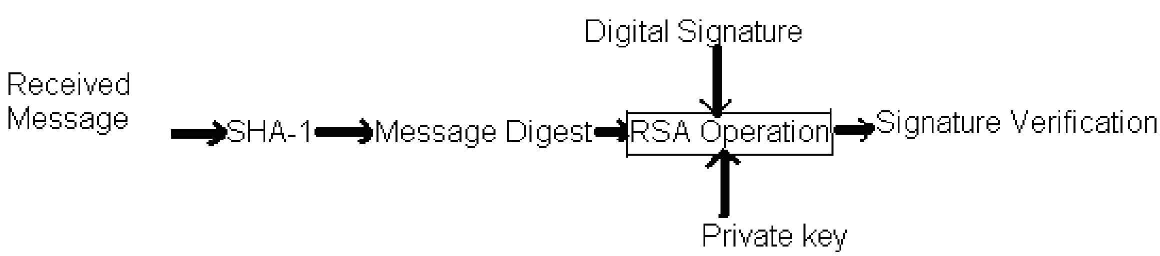


Fig.4.2 Signature Verification

2 –Encryption

A second element in digital signature is encryption. The most common encryption technology used is known as asymmetric cryptography, or public key cryptography

(RSA). In public key cryptography, a public key and private key are created. The private key is known only to the owner, while the public key can be distributed publicly. If the message is encrypted using the public key, then only the private key will be able to decrypt it. The reverse is also true if the message is encrypted with the private key.

4.3. Application Areas of Digital signatures

Digital signatures are needed by people and organizations in many different sectors. For example, individuals might use digital signatures for**:**

- Their medical records (remote treatment, e-mail, etc.)
- Financial transactions (bank transfers etc.)
- Education (online courses, registration, etc.)

Businesses might use digital signatures for**:**

- Financial transactions (securities trading, etc.)
- Conduct business discussions (company mergers etc.)
- communicating trade secrets
- filing legal documents

The government might use digital signatures for**:**

- military information
- Voting (registration, online voting, etc.)
- air traffic control information

4.4 Secure Hash Algorithm (SHA -1)

The Secure Hash Algorithm (SHA) was developed by the National Institute of Standards and Technology (NIST) and published as a federal information processing standard (FIPS

PUB 180) in 1993; a revised version was issued as FIPS PUB 180-1 in 1995 and is generally referred to as SHA-l. The algorithm takes as input a message with a maximum length of less than 264 bits and produces as output a 160-bit message digest. The input is processed in 512- bit blocks. Fig depicts the overall processing of a message to produce a digest. The processing consists of the following steps:

Step 1: Append Padding bits. The message is padded so that its length is congruent to 448 modulo 512(length = 448 mod 512). That is, the length of the padded message is 64 bits less than multiple of 512 bits. Padding is always added, even if the message is already of the desired length. Thus, the number of padding bits is in the range of 1 to 512. The padding consists of a single I-bit followed by the necessary number of 0-bits.

Step 2: Append length. A block of 64 bits is appended to the message. This block is treated as an unsigned 64-bit integer (most significant byte first) and contains the length of the original message (before the padding). The outcome of the first two steps yields a message that is an integer multiple of 512 bits in length. In figure 4.2, the expanded message is represented as the sequence of 512-bit blocks Mo, MI. ..., ML, so that the total length of the expanded message is L * 512 bits. Equivalently, the result is a multiple of 16 32-bit words.

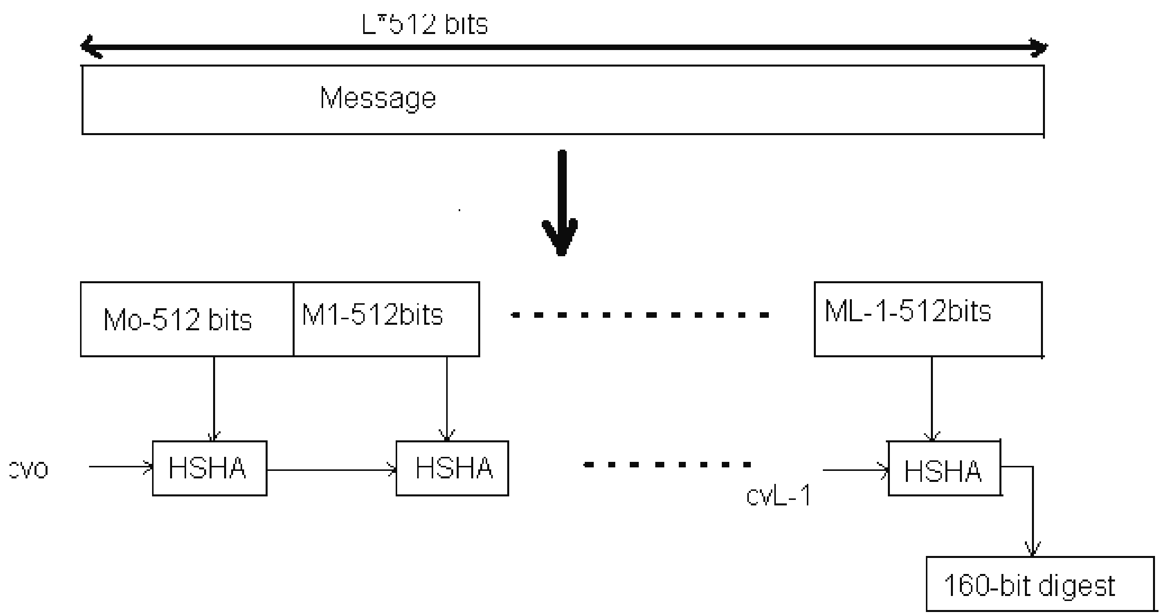


Fig . Message Digest Generation Using SHA-1

Step 3: Initialize array: An array of length 160 is used to hold intermediate and final results of the hash function. This array can be represented as five arrays of length 32 (A, B, C, D, and E). These arrays are initialized to the following 32-bit integers (hexadecimal values)**:**

A=67452301

B=EFCDAB89

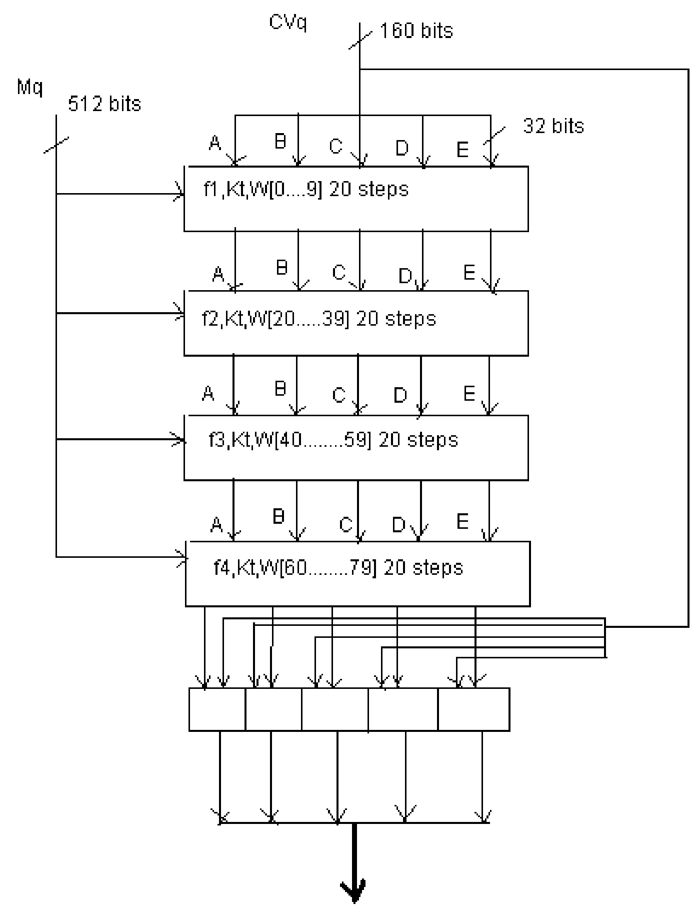
C=98BADCFE

D=10325476
E=C3D2E1FO

Step 4: Process message in 512-bit (16-word) blocks: The heart of the algorithm is a module that consists of four rounds of processing of 20 steps each. The logic is illustrated in Figure 4.3. The four rounds have similar structure but each uses different primitive logical functions which we refer to as fl, fl, f3, and f4 as shown in the table below. (See the detail in appendix) .Each round takes as input the current 512-bit block being processed and the 160-bit array value ABCDE and updates the contents of the array. Each round also makes use of an additive constant Kt, where 0<=t<=79 indicates one of the 80 steps across five rounds. In fact, only four distinct constants are used. The values, in hexadecimal, are as follows**:**

Step Number	**Hexadecimal**	**f_function**
0<=t<=19	Kt = 5A827999	f1 = (B&C) OR ((NOT B)&D)
20<=t<=39	Kt = 6ED9EBA1	f2 = B XOR C XOR D
40<=t<=59	Kt = 8FIBBCDC	f3 = (B &C)OR(B & D)OR(C & D)
60<=t<=79	Kt =CA62C1D64	f4 = B XOR C XOR D

Table 4.1



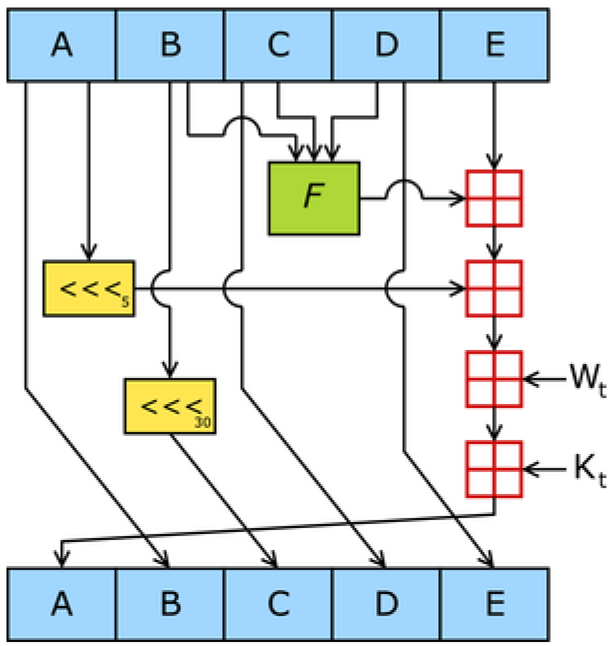
CV_{q+1} 160 bits
Fig 4.4. Processing of a single 512 –Bit Block

The output of the fourth round (eightieth step) is added to the input to the first round (CV_q) to produce CV $_{q+l}$. The addition is done independently for each of the five words in the array with each of the corresponding words in CV q, using addition modulo 2^{32}. One of the above 80 steps is shown in figure 4.4 below, where The output of the fourth round (eightieth step) is added to the input to the first round (CVq) to produce CV q+l. The addition is done independently for each of the five words in the array with each of the corresponding words in CV q, using addition modulo 2^{32}. One of the above 80 steps is shown in figure 4.4 below, where**:**

Wt (i) = (w (i-3) **xor** w (i-8) **xor** w (i-14) **xor** w (i-16)) **left rotate** 1 for 16 < i < 79

<<<5 and <<<30 implies **left rotate 5** and **left rotate 30** respectively.

Kt and **F** are as defined above.



Single step processing block

Fig 4.5. SHA-1

Step 5: Output: After all L 512-bit blocks have been processed, the output from the L th stage is the 160-bit message digest The SHA-l algorithm has the property that every bit of the hash code is a function of every bit in the input. The complex repetition of the basic function ft produces results that are well mixed; that is, it is unlikely that two messages chosen at random, even if they exhibit similar regularities, will have the same hash code. Unless there is some hidden weakness in SHA-l, which has not so far been published, the difficulty of coming up with two messages having the same message digest is on the order of 2^{80} operations, while the difficulty of finding a message with a given digest is on the order of 2^{160} operations.

Recall that when we introduced public key encryption earlier in this chapter, we said that it depends on two keys**:**

Public key ***:*** A key that is used for encrypting a secret message. Normally, the public key is widely published.

*Secret key (Private key)****:*** A key that is kept secret, for decrypting messages once they are received.

By using a little bit of mathematical gymnastics, you can run the public key algorithm in reverse. That is, you could encrypt messages with your secret key; these messages can then be decrypted by anyone who possesses your public key. Why would anyone want to do this? Each public key has one and only one matching secret key. If a particular public key can decrypt a message, you can be sure that the matching secret key was used to encrypt it. And that is how digital signatures work.

When you apply your secret key to a message, you are signing it. By using your secret key and the message digest function, you are able to calculate a digital signature for the message you're sending. In principle, a public key algorithm could be used without a message digest algorithm: we could encrypt the whole message with our private key. However, every public key algorithm in use requires a large amount of processor time to encrypt even moderate-size inputs. Thus, to sign a multi-megabyte file might take hours or days if we only used the public key encryption algorithm.

Instead, we use a fast message digest algorithm to calculate a hash value, and then we sign that small hash value with our secret key. When you, the recipient, get that small value, you can decrypt the hash value using our public key. You can also recreate the hash value from the input. If those two values match, you are assured that you got the same file we signed.

The most common digital signature in use today is the combination of the SHA-l message digest algorithm and the RSA public key encryption mechanism.

Chapter-4
Message Authentication and Hash Function

Encryption protects against passive attack (eavesdropping). A different requirement is to protect against active attacks (falsification of data and transactions). Protection against such attacks is known as message authentication. A message file, document or other collection of data is said to be authentic when it is genuine and came from its alleged source. Message authentication is a procedure that allows communicating parties to verify that received messages are authentic. The two important aspects are to verify that the contents of the message have not been altered and that the source is authentic

4.1 .Digital signatures

Digital signatures are an analog of handwritten signatures. *Digital signature* can reference the same definition used for electronic signatures `electronic signature' means an electronic sound, symbol, or process, attached to or logically associated with a contract or other record and executed or adopted by a person with the intent to sign the record .Handwritten signatures are based on the physically idiosyncratic way of signing one's name. But they can be easily forged. A digital signature is a mathematically precise way of attaching one's identity to a message. It is much harder to forge than a handwritten signature, and the signed message cannot be altered without also invalidating the signature.

Digital signatures rely on public key cryptography. Public key cryptography uses two "keys". Take an ordinary plain-text message and apply one of the keys to it in an encryption process, and you end up with a scrambled or "encrypted" (or, in the current context, "signed") message. Apply the other key to the scrambled message in a decryption process, and you end up with the original plain-text message.

One of these two keys is *'public."* Everyone knows what this key is--or can look it up, much like a telephone number. The other key is *'private."* Only you know your private key. Signing (encrypting) something with your private key puts your personal stamp on it. No one else can do this, because they don't know your private key.

A digital signature is (most often) a message digest encrypted with someone's private key to certify the contents. (A *message digest (cryptographic hash code)* is nothing more than a number -a special number that is effectively a hash code produced by a function that is very difficult to reverse.). This process of encryption is called signing. Digital signatures can perform two different functions, both very important to the security of your system:

- *Integrity:* A digital signature indicates whether a file or a message has been modified.
- *Authentication:* A digital signature makes possible mathematically verifying the

name of the person who signed the message.

- A third function that is quite valuable in some contexts is called *non-repudiation.* Non- repudiation means that after you have signed and sent a message, you cannot later claim that you did not sign the original message. You cannot repudiate your signature, because the message was signed with your private key (which, presumably, no one else has).

The most common digital signature technology currently in use is the Digital Signature Standard (DSS), which was accepted by the US Government as the official technology for federal electronic communications. The DSS specifies the use of the Digital Signature Algorithm (DSA) and the Secure Hash Algorithm (SHA-l) to produce digital signatures.

4.2. Calculation of Digital Signatures

1- Hash

A digital signature is created by fingerprinting electronic data using a mathematical formula, known as a hash algorithm. The Secure Hash Algorithm, used in the DSS, is one of several different algorithms available. The algorithm produces a message digest or "hash" which is derived from, and unique to, the data. Consequently, the digest is unique to a particular document and cannot be duplicated. When the digest is attached to the document, the document has been "signed". Reciepients can later use the digest to verify that the data have not been altered during transmission.

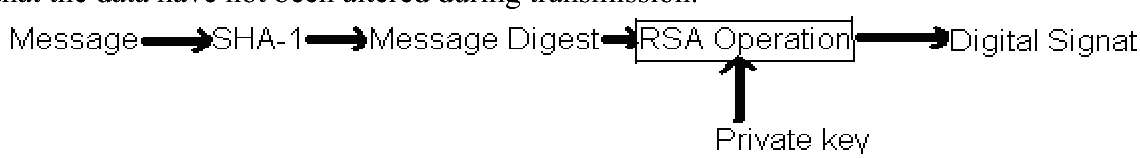


Fig.4.1 Signature Generation

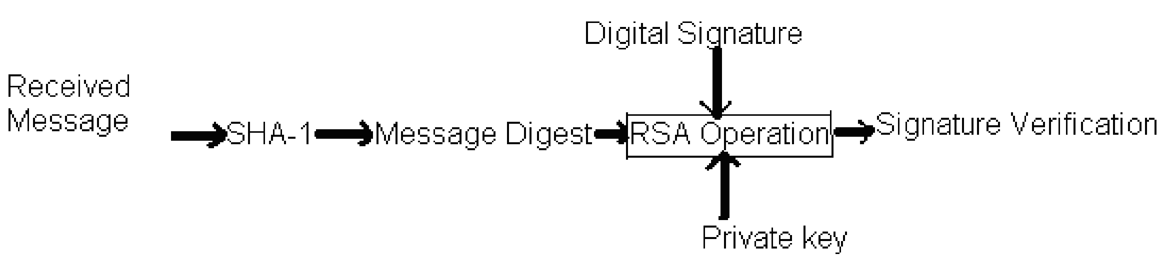


Fig.4.2 Signature Verification

2 –Encryption

A second element in digital signature is encryption. The most common encryption technology used is known as asymmetric cryptography, or public key cryptography (RSA). In public key cryptography, a public key and private key are created. The private key is known only to the owner, while the public key can be distributed publicly. If the message is encrypted using the public key, then only the private key will be able to decrypt it. The reverse is also true if the message is encrypted with the private key.

4.3. Application Areas of Digital signatures

Digital signatures are needed by people and organizations in many different sectors. For example, individuals might use digital signatures for**:**

- Their medical records (remote treatment, e-mail, etc.)
- Financial transactions (bank transfers etc.)
- Education (online courses, registration, etc.)

Businesses might use digital signatures for**:**

- Financial transactions (securities trading, etc.)
- Conduct business discussions (company mergers etc.)
- communicating trade secrets
- filing legal documents

The government might use digital signatures for**:**

- military information
- Voting (registration, online voting, etc.)
- air traffic control information

4.4 Secure Hash Algorithm (SHA -1)

The Secure Hash Algorithm (SHA) was developed by the National Institute of Standards and Technology (NIST) and published as a federal information processing standard (FIPS PUB 180) in 1993; a revised version was issued as FIPS PUB 180-1 in 1995 and is generally referred to as SHA-l. The algorithm takes as input a message with a maximum length of less than 264 bits and produces as output a 160-bit message digest. The input is processed in 512- bit blocks. Fig depicts the overall processing of a message to produce a digest. The processing consists of the following steps:

Step 1: Append Padding bits. The message is padded so that its length is congruent to 448 modulo 512(length = 448 mod 512). That is, the length of the padded message is 64 bits less than multiple of 512 bits. Padding is always added, even if the message is already of the desired length. Thus, the number of padding bits is in the range of 1 to 512. The padding consists of a single I-bit followed by the necessary number of 0-bits.

Step 2: Append length. A block of 64 bits is appended to the message. This block is treated as an unsigned 64-bit integer (most significant byte first) and contains the length of the original message (before the padding). The outcome of the first two steps yields a message that is an integer multiple of 512 bits in length. In figure 4.2, the expanded message is represented as the sequence of 512-bit blocks Mo, MI. ..., ML, so that the total length of the expanded message is L * 512 bits. Equivalently, the result is a multiple of 16 32-bit words.

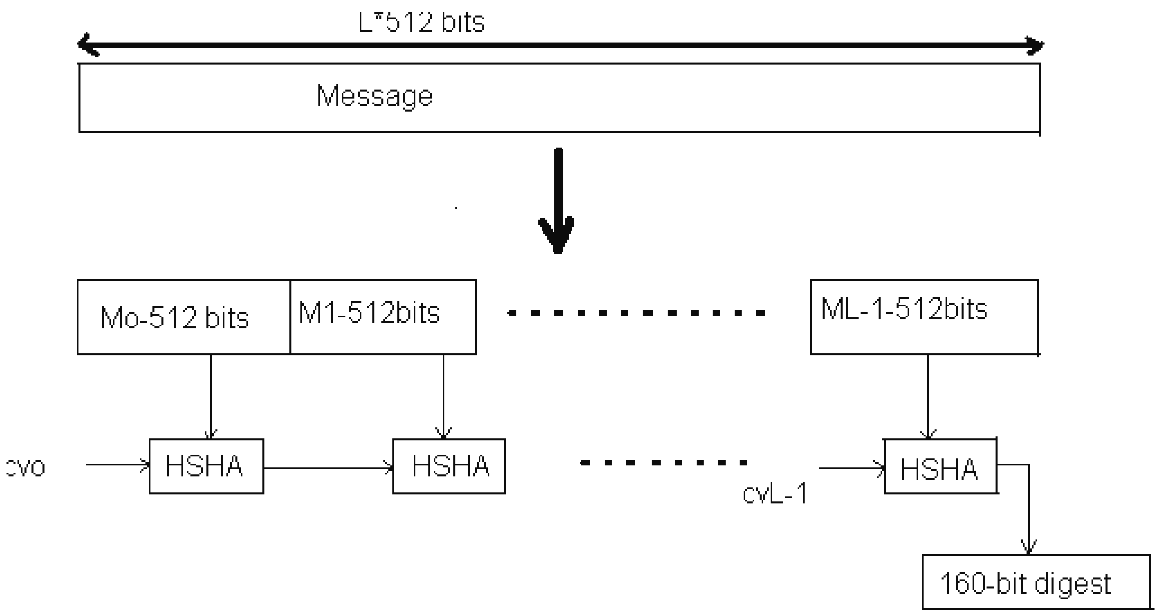


Fig . Message Digest Generation Using SHA-1

Step 3: Initialize array: An array of length 160 is used to hold intermediate and final results of the hash function. This array can be represented as five arrays of length 32 (A, B, C, D, and E). These arrays are initialized to the following 32-bit integers (hexadecimal values)**:**

A=67452301

B=EFCDAB89

C=98BADCFE

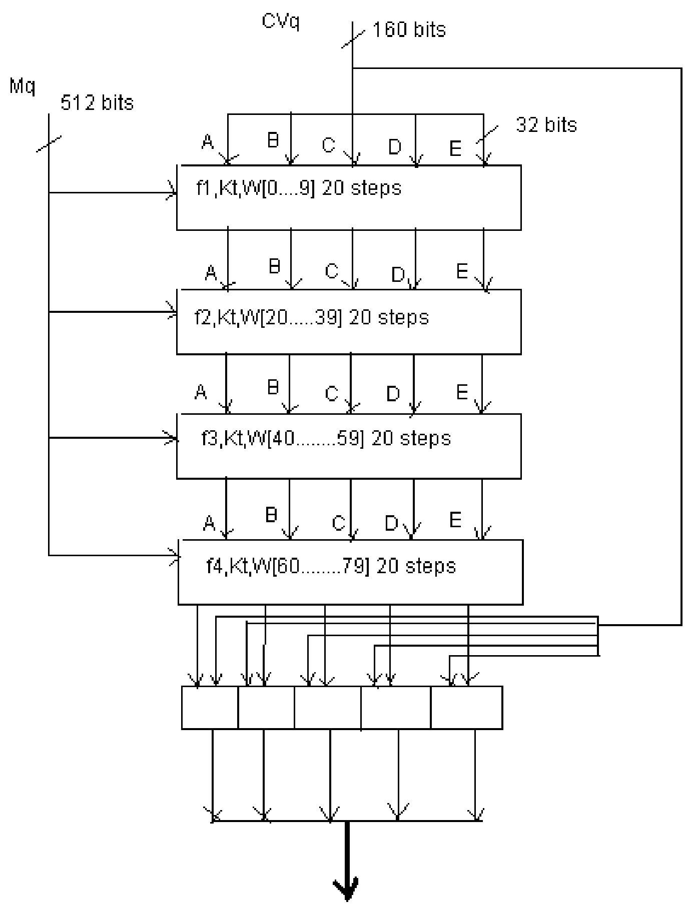
D=10325476

E=C3D2E1FO

Step 4: Process message in 512-bit (16-word) blocks: The heart of the algorithm is a module that consists of four rounds of processing of 20 steps each. The logic is illustrated in Figure 4.3. The four rounds have similar structure but each uses different primitive logical functions which we refer to as fl, fl, f3, and f4 as shown in the table below. (See the detail in appendix) .Each round takes as input the current 512-bit block being processed and the 160-bit array value ABCDE and updates the contents of the array. Each round also makes use of an additive constant Kt, where 0<=t<=79 indicates one of the 80 steps across five rounds. In fact, only four distinct constants are used. The values, in hexadecimal, are as follows**:**

Step Number	**Hexadecimal**	**f_function**
0<=t<=19	Kt = 5A827999	f1 = (B&C) OR ((NOT B)&D)
20<=t<=39	Kt = 6ED9EBA1	f2 = B XOR C XOR D
40<=t<=59	Kt = 8FIBBCDC	f3 = (B &C)OR(B & D)OR(C & D)
60<=t<=79	Kt =CA62C1D64	f4 = B XOR C XOR D

Table 4.1



CV_{q+1} 160 bits

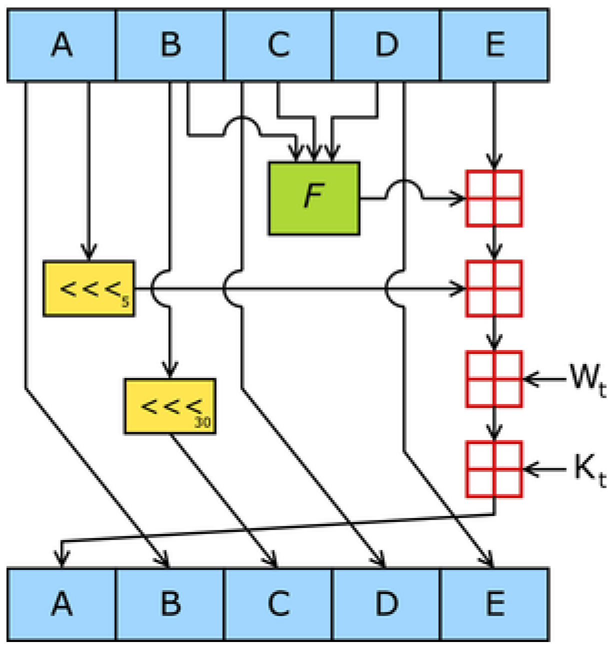
Fig 4.4. Processing of a single 512 –Bit Block

The output of the fourth round (eightieth step) is added to the input to the first round (CV_q) to produce CV $_{q+l}$. The addition is done independently for each of the five words in the array with each of the corresponding words in CV q, using addition modulo 2^{32}. One of the above 80 steps is shown in figure 4.4 below, where The output of the fourth round (eightieth step) is added to the input to the first round (CVq) to produce CV q+l. The addition is done independently for each of the five words in the array with each of the corresponding words in CV q, using addition modulo 2^{32}. One of the above 80 steps is shown in figure 4.4 below, where**:**

Wt (i) = (w (i-3) **xor** w (i-8) **xor** w (i-14) **xor** w (i-16)) **left rotate** 1 for 16 < i < 79

<<<5 and <<<30 implies **left rotate 5** and **left rotate 30** respectively.

Kt and **F** are as defined above.



Single step processing block
Fig 4.5. SHA-1

Step 5: Output: After all L 512-bit blocks have been processed, the output from the L th stage is the 160-bit message digest The SHA-l algorithm has the property that every bit of the hash code is a function of every bit in the input. The complex repetition of the basic function ft produces results that are well mixed; that is, it is unlikely that two messages chosen at random, even if they exhibit similar regularities, will have the same hash code. Unless there is some hidden weakness in SHA-l, which has not so far been published, the difficulty of coming up with two messages having the same message digest is on the order of 2^{80} operations, while the difficulty of finding a message with a given digest is on the order of 2^{160} operations.

Recall that when we introduced public key encryption earlier in this chapter, we said that it depends on two keys**:**

Public key ***:*** A key that is used for encrypting a secret message. Normally, the public key is widely published.

*Secret key (Private key)****:*** A key that is kept secret, for decrypting messages once they are received.

By using a little bit of mathematical gymnastics, you can run the public key algorithm in reverse. That is, you could encrypt messages with your secret key; these messages can then be decrypted by anyone who possesses your public key. Why would anyone want to do this? Each public key has one and only one matching secret key. If a particular public key can decrypt a message, you can be sure that the matching secret key was used to encrypt it. And that is how digital signatures work.

When you apply your secret key to a message, you are signing it. By using your secret key and the message digest function, you are able to calculate a digital signature for the message you're sending. In principle, a public key algorithm could be used without a message digest algorithm: we could encrypt the whole message with our private key. However, every public key algorithm in use requires a large amount of processor time to encrypt even moderate-size inputs. Thus, to sign a multi-megabyte file might take hours or days if we only used the public key encryption algorithm.

Instead, we use a fast message digest algorithm to calculate a hash value, and then we sign that small hash value with our secret key. When you, the recipient, get that small value, you can decrypt the hash value using our public key. You can also recreate the hash value from the input. If those two values match, you are assured that you got the same file we signed.

The most common digital signature in use today is the combination of the SHA-l message digest algorithm and the RSA public key encryption mechanism.

Chapter-5

Implementation and Sample Outputs of Image Cryptography

5.1. Preliminary steps for image encryption

5.1.1 .Generation of two prime numbers, p and q;

Two prime numbers used for RSA key generation are appropriately selected from among many prime numbers generated by a simple program in a delimited range.

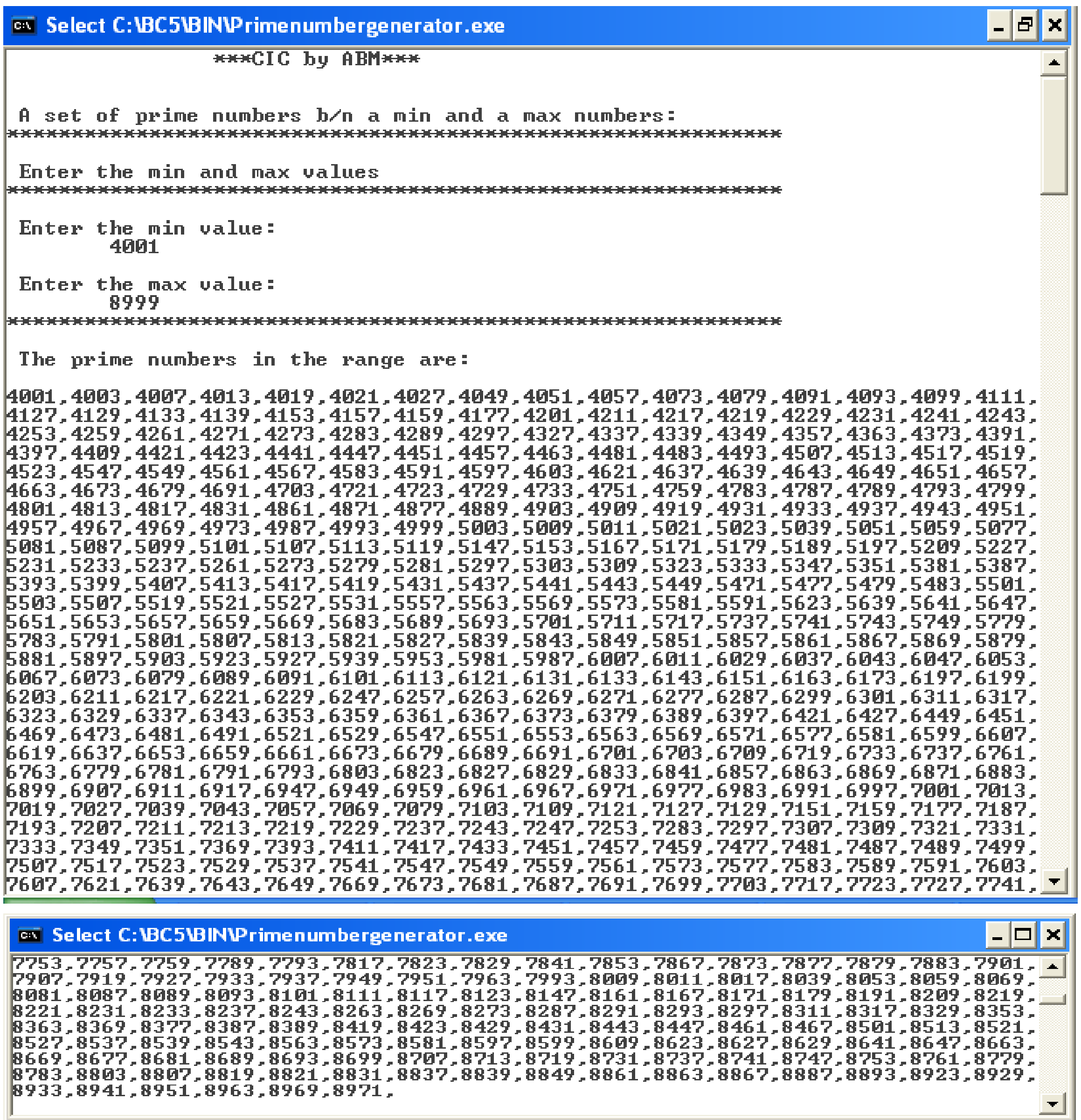


Fig 5.1. Two-prime number generation

From the above set of prime numbers in the range *4001* to *8999*, two prime numbers ***p=7923*** and ***q=8731*** are chosen as input for the next process. That's, for RSA key generation. They are used to calculate n, the modulus for encryption and the quantity, Φ(n), which is referred to as the Euler totient of n which are in turn used to calculate the public and private keys as depicted in the next step.

5.1.2. Key generation

The Euclidean algorithm for key generation is implemented using a C++ code as follows. The above generated two prime numbers, p and q, are input to this algorithm

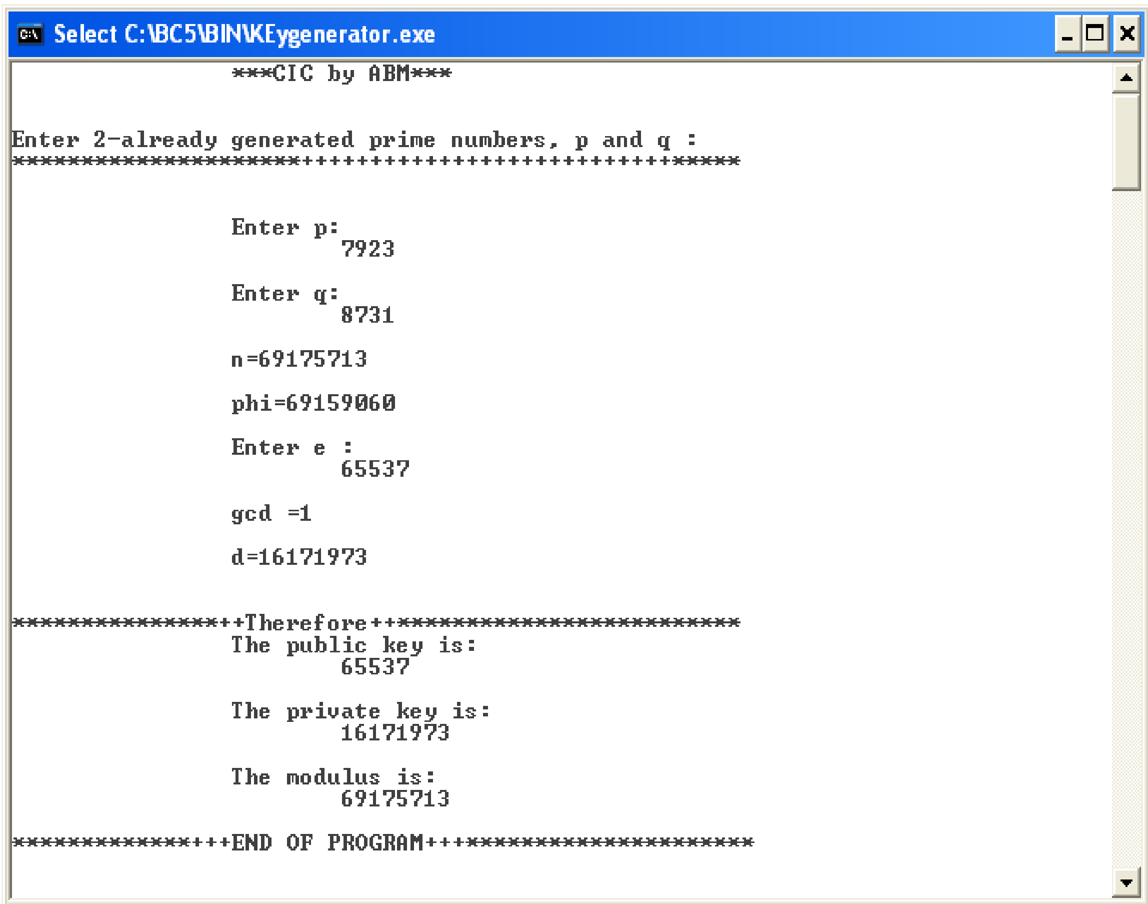


Fig 5.2 RSA key generation

Therefore, the outputs of the above program, the public key, the private key and the modulus of encryption serves as inputs to the next step, DES key encryption.

5.2. DES key encryption

We selected an 8-digit DES key, ***15040633***, for the encryption of the image. To make the communication more secure, this key is encrypted using RSA algorithm and sent along with the encrypted image to the receiver who's in the know of the public key. The encrypted DES key is therefore displayed as follows:

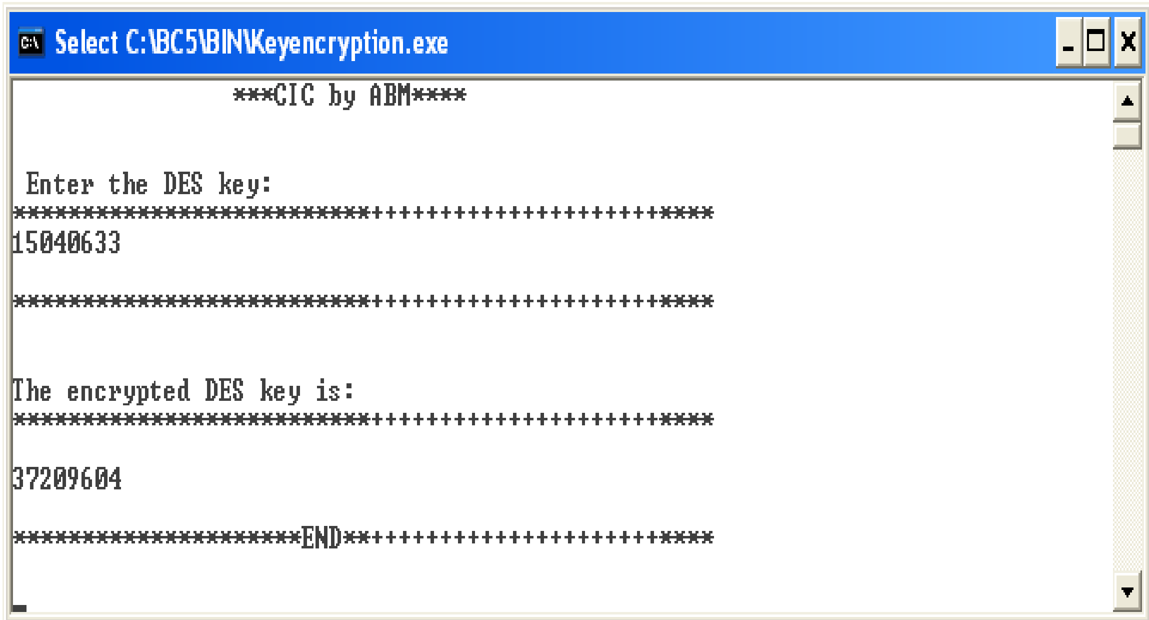


Fig 5.3 DES key encryption

5.3. DES key decryption

The encrypted DES key is decrypted at the receiving point using the reverse of RSA algorithm.

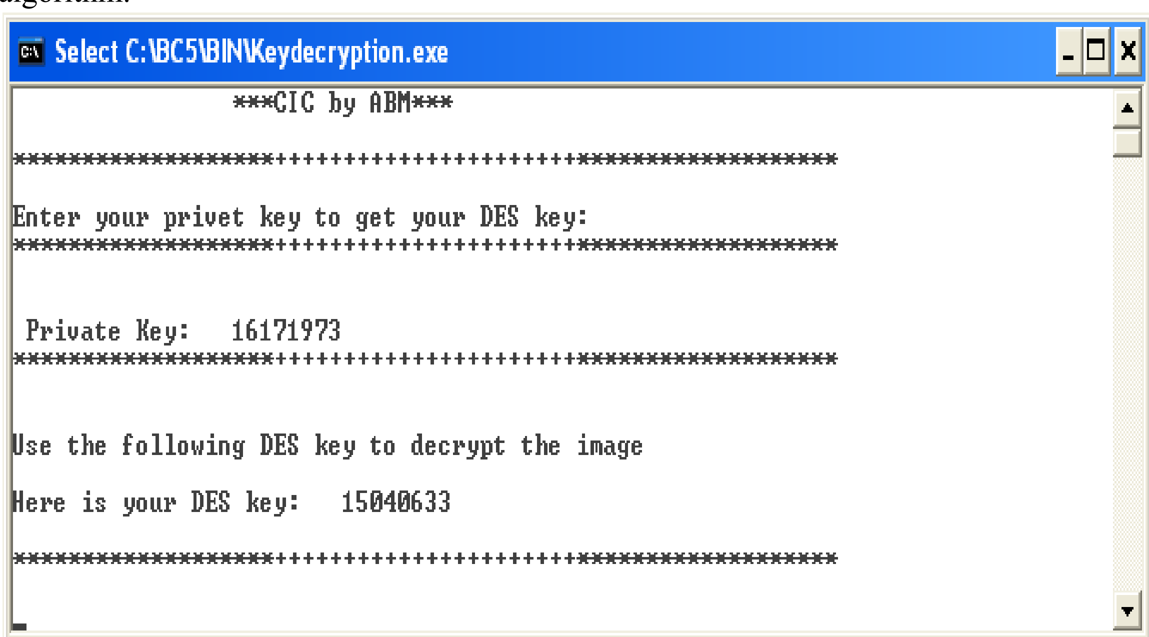


Fig 5.4 DES key decryption

5.4 .Reading pixel values of the image to be encrypted

The following image is of pixel size 640 by480. It is an RGB (real) image.

Fig 5.5 input image

The above image was converted into an intensity (gray scale) image having color intensity values that range from 0 (black) to 255(white) for the sake of simplicity while reading the pixel values for it is a two dimensional image (n by m).

Fig 5.6 gray scale input image

As depicted in the above fig the array size of image is 640 by *480* which is too large to be handled by our pc (the software we run the program). Consequently, we resized it to the array size that our pc can handle. It was reduced to *32* by *32* array size which is easily handled by our pc. Then the resized gray scale image looks like as in the fig7 on which encryption and decryption processes are to be performed.

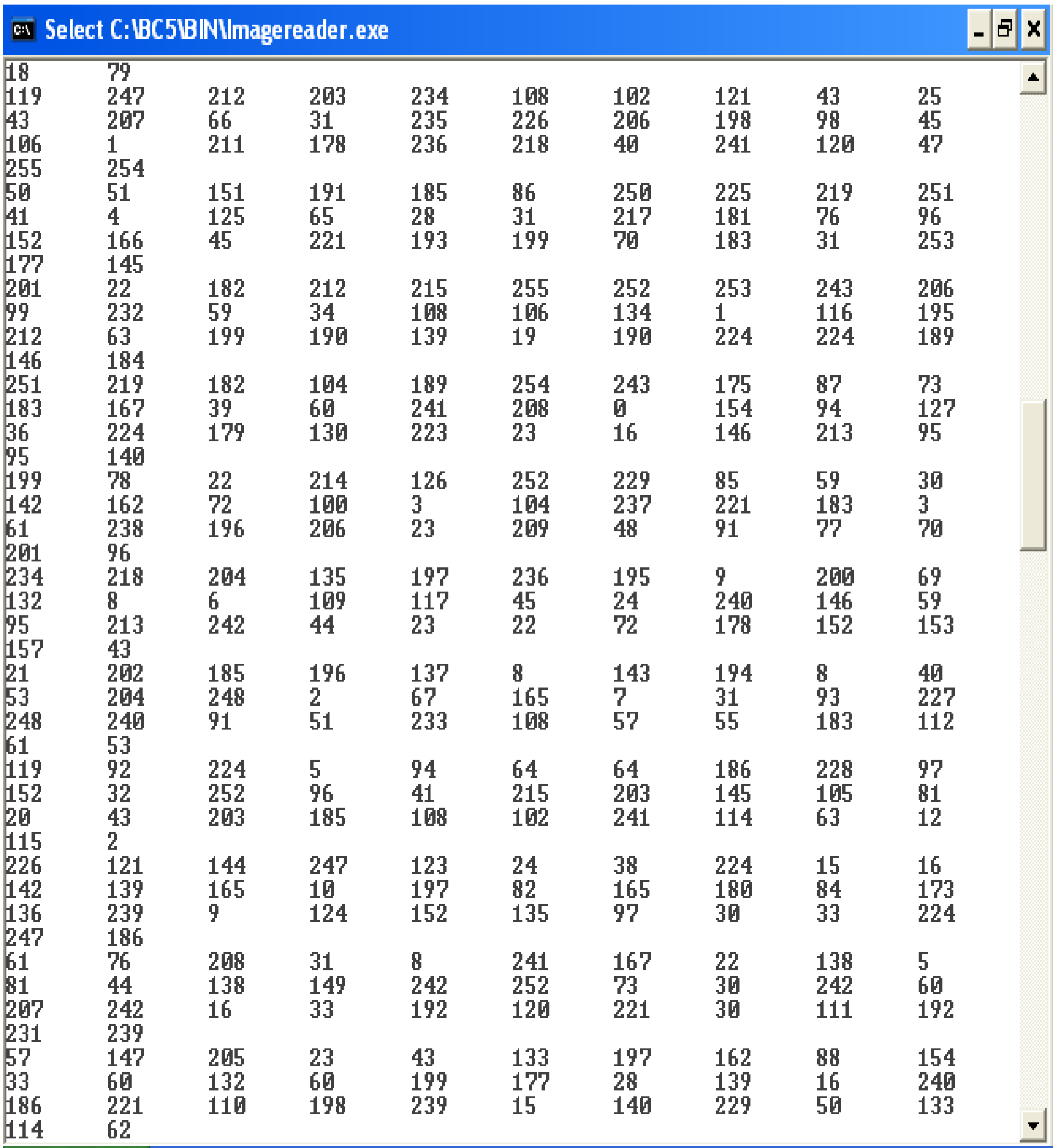
Fig 5.7 resized gray scale image

The pixel values of the resized gray scale image are read using a C++ code and produced the following output.

```
Select C:\BC5\BIN\Imagereader.exe

Enter the image dimensions:
Enter the width,W:
        32

Enter the height,H:
        32

Enter the proper path of the saved image:

        c:\\bett.png
137     80      78      71      13      10      26      10      0       0
0       13      73      72      68      82      0       0       0       32
0       0       0       32      8       0       0       0       0       86
17      37
40      0       0       0       9       112     72      89      115     0
0       11      18      0       0       11      18      1       210     221
126     252     0       0       3       209     73      68      65      84
120     156
53      143     207     83      27      85      0       128     223     255
145     150     146     100     243     131     236     219     125     217
188     151     54      100     96      160     20      164     63      36
54      104
169     180     64      105     37      77      132     36      36      217
132     20      138     69      41      50      82      157     142     76
177     246     96      103     236     193     41      157     170     147
10      45
36      217     205     102     127     191     205     201     241     232
140     142     51      189     244     228     56      29      199     179
7       235     249     251     14      223     7       94      28      214
235     77
73      146     100     73      146     164     189     197     219     7
82      163     209     108     54      154     77      73      106     212
15      15      14      64      189     222     148     100     185     165
180     100
89      254     41      25      37      195     183     158     75      146
36      201     45      89      150     155     245     122     29      52
36      169     165     40      109     181     221     86      106     9
66      8
33      3       43      123     114     171     165     180     149     150
220     108     52      128     36      183     148     182     170     106
154     186     159     192     56      18      193     152     144     254
234     158
170     170     170     170     200     82      19      188     193     186
246     60      65      194     16      98      130     49      22      72
```

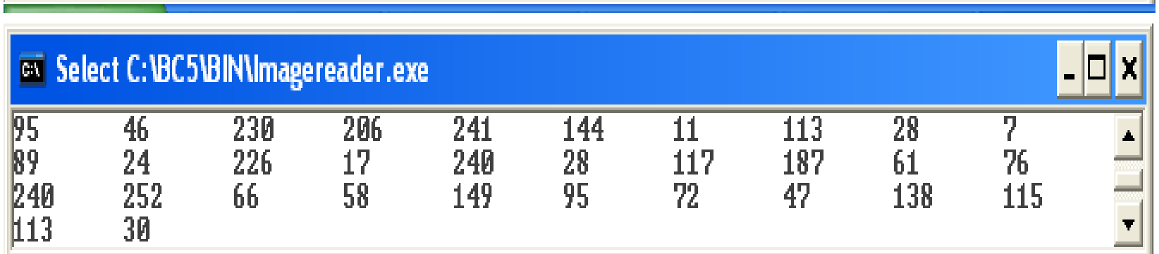
Select C:\BC5\BIN\Imagereader.exe

95	249	63	165	173	0	69	213	52	93
55	244								
253	115	4	177	16	97	66	8	193	2
38	241	226	51	93	83	85	21	180	53
221	48	76	107	255	12	70	144	19	48
142	18								
1	33	46	140	9	137	231	30	105	170
6	84	77	55	76	203	42	245	18	14
97	66	72	148	68	16	100	145	64	72
244	120								
236	75	93	7	154	110	152	150	109	219
79	223	239	27	75	92	152	60	63	16
197	2	130	33	68	8	25	17	15	116
29	232								
134	105	217	212	113	204	177	209	204	218
237	213	114	118	18	35	30	194	48	142
22	228	187	154	97	0	195	180	108	135
58	102								
105	104	106	122	227	227	242	86	41	125
162	203	7	81	228	244	119	84	45	24
166	9	12	211	182	169	227	124	45	12
93	157								
185	113	115	233	106	110	245	84	151	139
33	75	58	165	107	189	223	216	54	48
45	155	58	29	123	42	56	191	154	185
241	201								
250	246	253	149	222	110	23	83	161	157
78	231	222	224	19	74	129	105	81	234
56	102	33	180	40	206	223	218	88	255
126	39								
131	93	221	204	128	222	233	116	86	250
159	81	10	76	203	113	156	78	99	26
142	95	188	82	169	150	239	174	38	185
158	163								
71	142	93	161	157	78	122	176	70	41
176	108	74	29	101	4	117	9	19	169
245	173	207	54	151	79	179	190	32	100
131	79								
59	116	186	255	128	82	96	218	14	117
30	143	162	96	23	201	108	239	124	90
152	124	7	6	134	93	193	174	249	67
249	221								
254	198	255	13	15	120	15	116	49	233
244	165	11	19	215	251	56	215	184	139
117	5	178	181	225	184	242	230	130	110



Select C:\BC5\BIN\Imagereader.exe

18	79								
119	247	212	203	234	108	102	121	43	25
43	207	66	31	235	226	206	198	98	45
106	1	211	178	236	218	40	241	120	47
255	254								
50	51	151	191	185	86	250	225	219	251
41	4	125	65	28	31	217	181	76	96
152	166	45	221	193	199	70	183	31	253
177	145								
201	22	182	212	215	255	252	253	243	206
99	232	59	34	108	106	134	1	116	195
212	63	199	190	139	19	190	224	224	189
146	184								
251	219	182	104	189	254	243	175	87	73
183	167	39	60	241	208	0	154	94	127
36	224	179	130	223	23	16	146	213	95
95	140								
199	78	22	214	126	252	229	85	59	30
142	162	72	100	3	104	237	221	183	3
61	238	196	206	23	209	48	91	77	70
201	96								
234	218	204	135	197	236	195	9	200	69
132	8	6	109	117	45	24	240	146	59
95	213	242	44	23	22	72	178	152	153
157	43								
21	202	185	196	137	8	143	194	8	40
53	204	248	2	67	165	7	31	93	227
248	240	91	51	233	108	57	55	183	112
61	53								
119	92	224	5	94	64	64	186	228	97
152	32	252	96	41	215	203	145	105	81
20	43	203	185	108	102	241	114	63	12
115	2								
226	121	144	247	123	24	38	224	15	16
142	139	165	10	197	82	165	180	84	173
136	239	9	124	152	135	97	30	33	224
247	186								
61	76	208	31	8	241	167	22	138	5
81	44	138	149	242	252	73	30	242	60
207	242	16	33	192	120	221	30	111	192
231	239								
57	147	205	23	43	133	197	162	88	154
33	60	132	60	199	177	28	139	16	240
186	221	110	198	239	15	140	229	50	133
114	62								



Select C:\BC5\BIN\Imagereader.exe

95	46	230	206	241	144	11	113	28	7
89	24	226	17	240	28	117	187	61	76
240	252	66	58	149	95	72	47	138	115
113	30								

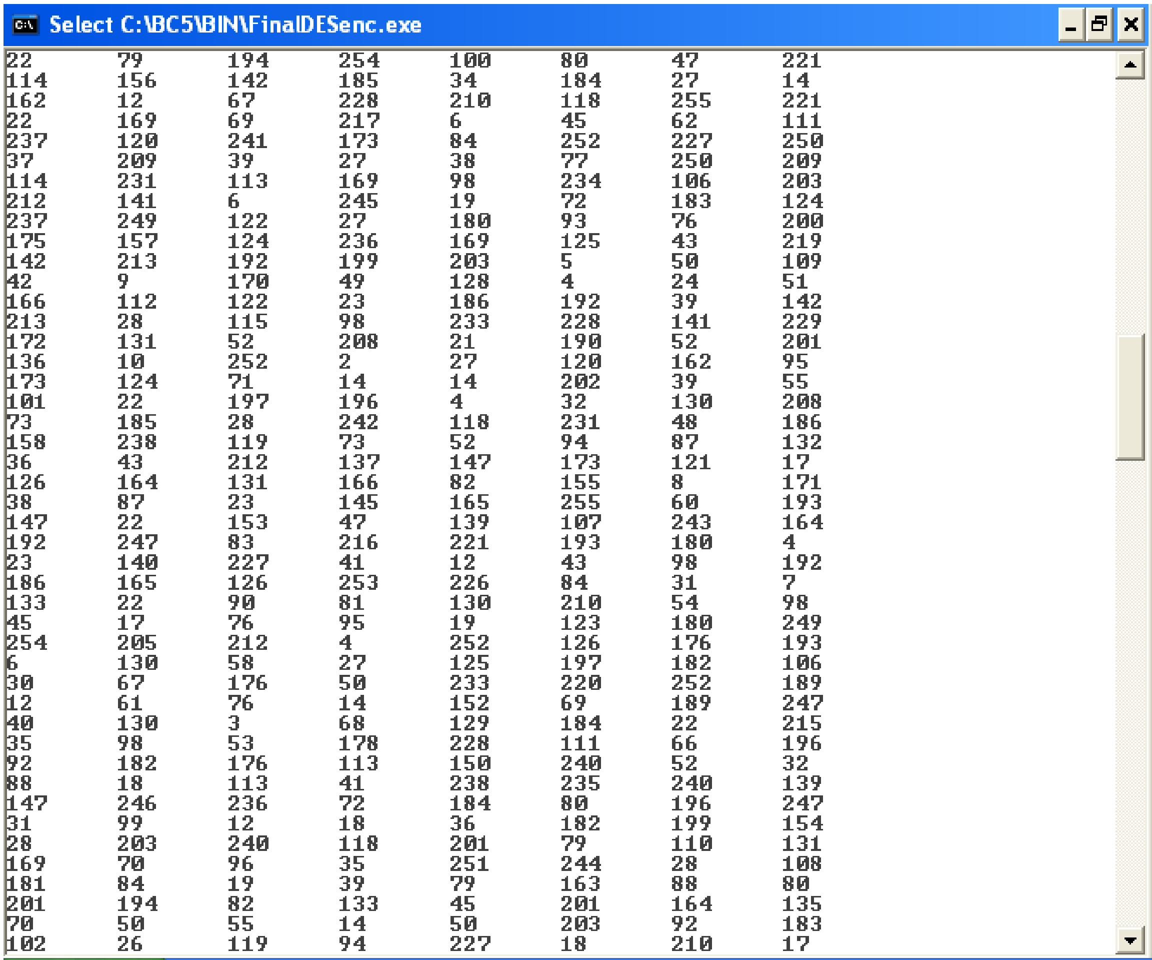
Fig 5.8 matrix of the pixel values of the resized gray scale original image

5.5. Encryption of the image using DES algorithm

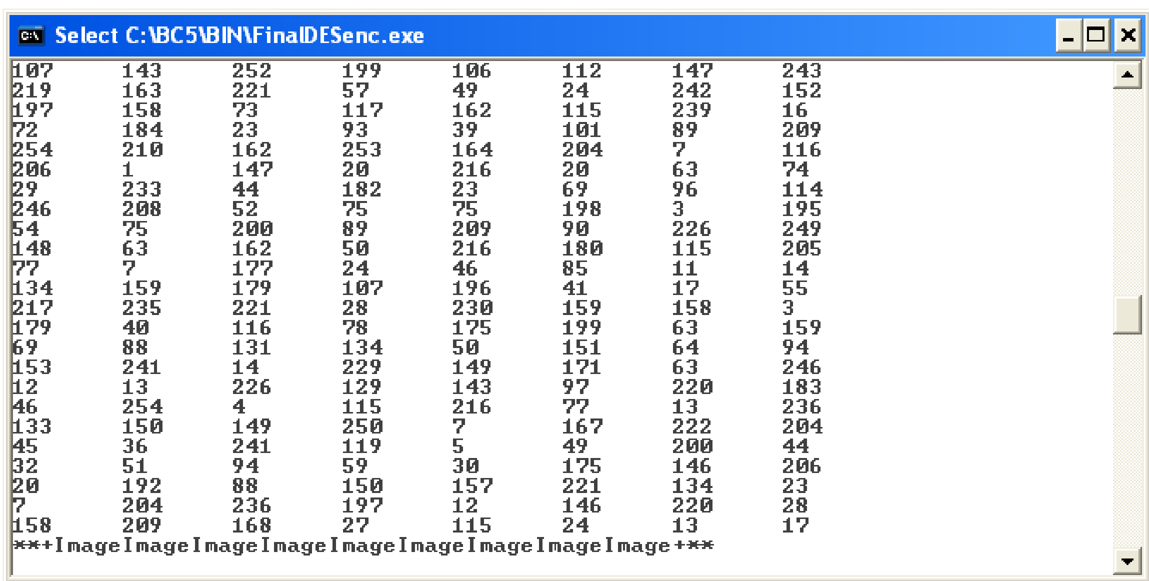
Each and every pixel value of the resized gray scale image is encrypted using DES algorithm and the DES key, ***15040633.***The encrypted pixel values of the input image in matrix form are displayed in the following screen.

```
Select C:\BC5\BIN\FinalDESenc.exe

Enter the image dimensions in pixel:
Enter the width,W:
        32

Enter the height,H:
        32

Enter the proper path of the saved image:

        c:\\bett.png

 Enter your 8-digit DES key:

key[1]=1
key[2]=5
key[3]=0
key[4]=4
key[5]=0
key[6]=6
key[7]=3
key[8]=3

 The cipher(encrypted) image in integer format is:

***+ImageImageImageImageImageImageImageImageImage+***

4       116     123     11      32      213     245     207
236     23      230     165     138     113     152     194
120     60      155     104     171     154     77      207
189     74      97      242     111     236     226     102
34      37      191     91      61      185     90      128
21      41      6       48      49      20      150     88
183     0       42      232     50      2       127     162
214     242     66      191     79      67      55      209
250     110     197     142     235     31      210     67
141     57      94      67      51      70      170     109
53      183     216     0       180     8       145     75
135     99      62      203     74      250     169     34
200     210     249     143     82      203     245     29
150     27      151     131     158     252     51      68
```

```
Select C:\BC5\BIN\FinalDESenc.exe
227     186     191     155     223     122     8       212
208     105     98      234     10      210     57      223
230     212     248     165     119     216     0       223
133     134     23      17      109     242     117     169
195     188     214     100     117     42      110     185
59      26      164     127     166     140     78      200
29      133     153     73      235     209     151     13
11      89      112     203     2       46      238     82
172     112     24      96      20      100     91      250
117     48      243     197     106     84      243     171
38      12      93      101     79      138     222     255
29      198     37      151     156     31      37      49
187     251     41      162     46      5       117     147
216     77      25      203     152     208     172     79
253     161     38      118     253     156     189     22
143     48      214     163     231     140     134     168
171     105     57      68      42      32      92      53
210     199     200     18      182     163     204     169
250     176     12      119     129     90      77      140
181     175     188     235     95      118     144     58
23      239     10      157     24      150     66      172
11      255     235     172     47      64      239     141
53      230     139     181     160     34      26      217
93      242     26      38      225     154     147     184
76      0       53      217     165     26      195     146
8       40      128     239     41      79      33      163
124     104     212     16      97      43      1       140
52      148     151     7       59      219     77      175
78      6       43      212     236     218     209     52
204     196     47      21      3       253     173     31
246     70      78      116     111     31      60      22
209     133     80      185     211     140     11      12
79      48      206     20      185     4       213     102
193     166     59      16      187     1       108     150
100     138     210     103     177     58      153     97
63      133     197     183     71      235     118     7
67      37      246     255     129     202     94      209
192     187     208     119     80      195     14      233
89      217     114     188     63      85      177     82
166     60      64      94      184     106     102     101
236     217     21      138     185     1       164     171
0       51      82      247     193     82      114     47
215     224     185     127     200     255     21      105
151     235     40      91      226     124     23      43
4       86      193     210     148     100     176     29
```



22	79	194	254	100	80	47	221
114	156	142	185	34	184	27	14
162	12	67	228	210	118	255	221
22	169	69	217	6	45	62	111
237	120	241	173	84	252	227	250
37	209	39	27	38	77	250	209
114	231	113	169	98	234	106	203
212	141	6	245	19	72	183	124
237	249	122	27	180	93	76	200
175	157	124	236	169	125	43	219
142	213	192	199	203	5	50	109
42	9	170	49	128	4	24	51
166	112	122	23	186	192	39	142
213	28	115	98	233	228	141	229
172	131	52	208	21	190	52	201
136	10	252	2	27	120	162	95
173	124	71	14	14	202	39	55
101	22	197	196	4	32	130	208
73	185	28	242	118	231	48	186
158	238	119	73	52	94	87	132
36	43	212	137	147	173	121	17
126	164	131	166	82	155	8	171
38	87	23	145	165	255	60	193
147	22	153	47	139	107	243	164
192	247	83	216	221	193	180	4
23	140	227	41	12	43	98	192
186	165	126	253	226	84	31	7
133	22	90	81	130	210	54	98
45	17	76	95	19	123	180	249
254	205	212	4	252	126	176	193
6	130	58	27	125	197	182	106
30	67	176	50	233	220	252	189
12	61	76	14	152	69	189	247
40	130	3	68	129	184	22	215
35	98	53	178	228	111	66	196
92	182	176	113	150	240	52	32
88	18	113	41	238	235	240	139
147	246	236	72	184	80	196	247
31	99	12	18	36	182	199	154
28	203	240	118	201	79	110	131
169	70	96	35	251	244	28	108
181	84	19	39	79	163	88	80
201	194	82	133	45	201	164	135
70	50	55	14	50	203	92	183
102	26	119	94	227	18	210	17



107	143	252	199	106	112	147	243
219	163	221	57	49	24	242	152
197	158	73	117	162	115	239	16
72	184	23	93	39	101	89	209
254	210	162	253	164	204	7	116
206	1	147	20	216	20	63	74
29	233	44	182	23	69	96	114
246	208	52	75	75	198	3	195
54	75	200	89	209	90	226	249
148	63	162	50	216	180	115	205
77	7	177	24	46	85	11	14
134	159	179	107	196	41	17	55
217	235	221	28	230	159	158	3
179	40	116	78	175	199	63	159
69	88	131	134	50	151	64	94
153	241	14	229	149	171	63	246
12	13	226	129	143	97	220	183
46	254	4	115	216	77	13	236
133	150	149	250	7	167	222	204
45	36	241	119	5	49	200	44
32	51	94	59	30	175	146	206
20	192	88	150	157	221	134	23
7	204	236	197	12	146	220	28
158	209	168	27	115	24	13	17

+ImageImageImageImageImageImageImageImageImage+

Fig 5.9 matrix of the pixel values of the encrypted image

The above matrix of pixel values of the encrypted image are transformed to cipher image that they represent using the following matlab command.

figure;

colormap(gray(256));

Image (pixel)

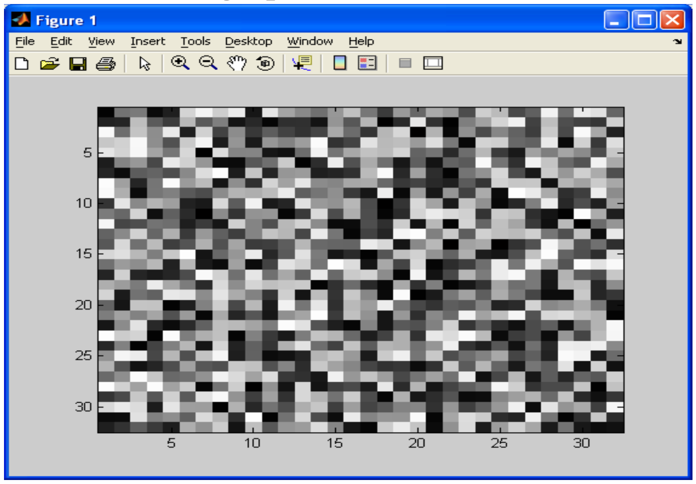


Fig 5.10 encrypted image

5.6. Decryption of the encrypted image

The encrypted image is decrypted by applying the inverse process of encryption and the DES key which was used during the encryption process. That is, ***15040633.***

```
Select C:\BC5\BIN\FinalDESdec.exe
                  ***CIC by ABM***

 Enter your 8-digit DES key :

***********************++++++++++++++++++++++++++++++++++++*****++*****

key[1]=1
key[2]=5
key[3]=0
key[4]=4
key[5]=0
key[6]=6
key[7]=3
key[8]=3


***********************++++++++++++++++++++++++++++++++++++*****++*****


 The decrypted image is:

+++++++++++++++++++++++Image+++++++++++++++++++++++++++++++++++++++++++++++++
137     80      78      71      13      10      26      10      0       0
0       13      73      72      68      82      0       0       0       32
0       0       0       32      8       0       0       0       0       86
17      37      40      0       0       0       9       112     72      89
115     0       0       11      18      0       0       11      18      1
210     221     126     252     0       0       3       209     73      68
65      84      120     156     53      143     207     83      27      85
0       128     223     255     145     150     146     100     243     131
236     219     125     217     188     151     54      100     96      160
20      164     63      36      54      104     169     180     64      105
37      77      132     36      36      217     132     20      138     69
41      50      82      157     142     76      177     246     96      103
236     193     41      157     170     147     10      45      36      217
205     102     127     191     205     201     241     232     140     142
51      189     244     228     56      29      199     179     7       235
249     251     14      223     7       94      28      214     235     77
73      146     100     73      146     164     189     197     219     7
82      163     209     108     54      154     77      73      106     212
15      15      14      64      189     222     148     100     185     165
180     100     89      254     41      25      37      195     183     158
75      146     36      201     45      89      150     155     245     122
29      52      36      169     165     40      109     181     221     86
```

```
Select C:\BC5\BIN\FinalDESdec.exe
106     9       66      8       33      3       43      123     114     171
165     180     149     150     220     108     52      128     36      183
148     182     170     106     154     186     159     192     56      18
193     152     144     254     234     158     170     170     170     170
200     82      19      188     193     186     246     60      65      194
16      98      130     49      22      72      95      249     63      165
173     0       69      213     52      93      55      244     253     115
4       177     16      97      66      8       193     2       38      241
226     51      93      83      85      21      180     53      221     48
76      107     255     12      70      144     19      48      142     18
1       33      46      140     9       137     231     30      105     170
6       84      77      55      76      203     42      245     18      14
97      66      72      148     68      16      100     145     64      72
244     120     236     75      93      7       154     110     152     150
109     219     79      223     239     27      75      92      152     60
63      16      197     2       130     33      68      8       25      17
15      116     29      232     134     105     217     212     113     204
177     209     204     218     237     213     114     118     18      35
30      194     48      142     22      228     187     154     97      0
195     180     108     135     58      102     105     104     106     122
227     227     242     86      41      125     162     203     7       81
228     244     119     84      45      24      166     9       12      211
182     169     227     124     45      12      93      157     185     113
115     233     106     110     245     84      151     139     33      75
58      165     107     189     223     216     54      48      45      155
58      29      123     42      56      191     154     185     241     201
250     246     253     149     222     110     23      83      161     157
78      231     222     224     19      74      129     105     81      234
56      102     33      180     40      206     223     218     88      255
126     39      131     93      221     204     128     222     233     116
86      250     159     81      10      76      203     113     156     78
99      26      142     95      188     82      169     150     239     174
38      185     158     163     71      142     93      161     157     78
122     176     70      41      176     108     74      29      101     4
117     9       19      169     245     173     207     54      151     79
179     190     32      100     131     79      59      116     186     255
128     82      96      218     14      117     30      143     162     96
23      201     108     239     124     90      152     124     7       6
134     93      193     174     249     67      249     221     254     198
255     13      15      120     15      116     49      233     244     165
11      19      215     251     56      215     184     139     117     5
178     181     225     184     242     230     130     110     18      79
119     247     212     203     234     108     102     121     43      25
43      207     66      31      235     226     206     198     98      45
106     1       211     178     236     218     40      241     120     47
```

```
Select C:\BC5\BIN\FinalDESdec.exe
255   254   50    51    151   191   185   86    250   225
219   251   41    4     125   65    28    31    217   181
76    96    152   166   45    221   193   199   70    183
31    253   177   145   201   22    182   212   215   255
252   253   243   206   99    232   59    34    108   106
134   1     116   195   212   63    199   190   139   19
190   224   224   189   146   184   251   219   182   104
189   254   243   175   87    73    183   167   39    60
241   208   0     154   94    127   36    224   179   130
223   23    16    146   213   95    95    140   199   78
22    214   126   252   229   85    59    30    142   162
72    100   3     104   237   221   183   3     61    238
196   206   23    209   48    91    77    70    201   96
234   218   204   135   197   236   195   9     200   69
132   8     6     109   117   45    24    240   146   59
95    213   242   44    23    22    72    178   152   153
157   43    21    202   185   196   137   8     143   194
8     40    53    204   248   2     67    165   7     31
93    227   248   240   91    51    233   108   57    55
183   112   61    53    119   92    224   5     94    64
64    186   228   97    152   32    252   96    41    215
203   145   105   81    20    43    203   185   108   102
241   114   63    12    115   2     226   121   144   247
123   24    38    224   15    16    142   139   165   10
197   82    165   180   84    173   136   239   9     124
152   135   97    30    33    224   247   186   61    76
208   31    8     241   167   22    138   5     81    44
138   149   242   252   73    30    242   60    207   242
16    33    192   120   221   30    111   192   231   239
57    147   205   23    43    133   197   162   88    154
33    60    132   60    199   177   28    139   16    240
186   221   110   198   239   15    140   229   50    133
114   62    95    46    230   206   241   144   11    113
28    7     89    24    226   17    240   28    117   187
61    76    240   252   66    58    149   95    72    47
138   115   113   30
++++++++++++++++++++++Image++++++++++++++++++++++++++++++++++++++++++++++++++
```

Fig 5.11 decrypted pixel values of the image

As can be seen from the above display, the decrypted pixel values are exactly the same as the pixel values of the original gray scale image.

Fig 5.12 decrypted image

The decrypted image is exactly the same as the compressed gray scale input image.

To obtain the original normal size gray scale image, the decrypted image can be enlarged (Scaled up) by the inverse of the factor that was used to scale it down before encryption to suit the pc capacity (array size).

5.7. Calculation of Message Digest of the original image.

A message digest is calculated from the original image. It is a function of the image,

MD=f(image)

It serves as proof (authentication) whether the image comes from the right sender. In other words, it gives a proof whether the image was modified during transmission or not.

```
Select C:\BC5\BIN\MD32by32.exe
                ***CIC by ABM***

 The message digest of the original image is:

*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*

2961462722201824478343761605721989542392335644140

*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*
```

Fig 5.13 message digest of the original image

5.8. Encryption of the message digests of the original image.

The message digest produced from the input image is encrypted before transmission for the purpose of strengthening security.

Fig 5.14 encryption of the message digests of the original gray scale image

5.9. Calculation of the message digests of the decrypted image.

The message digest of the decrypted image is produced in a similar fashion as that of the input image, and it is the same as the message digest of the original image proving that the image has reached its destination without any kind of hacking, or modification.

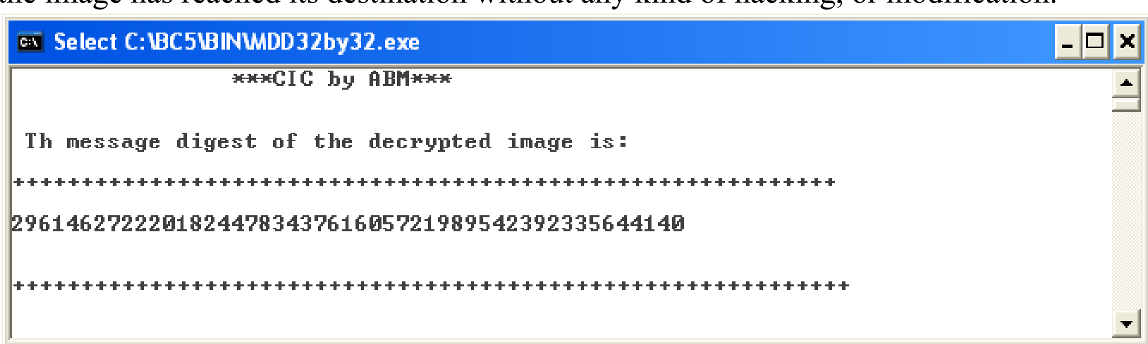


Fig 5.15 Message digest of the decrypted image

5.10 .Decryption of the encrypted message digest using the inverse process of RSA, and checking and comparing the input and the output message digests of the image.

Here the message digest of the decrypted image is calculated and compared with the decrypted message digest of the original image.

- ✔ If the image's been not modified during transmission from the sender to the receiver, the following message is displayed:

Fig 5.16 comparison of the message digests of the original image and the decrypted image.

✔ If the image has been modified during transmission by unauthorized parties, the sample output looks like displayed in screen the below.

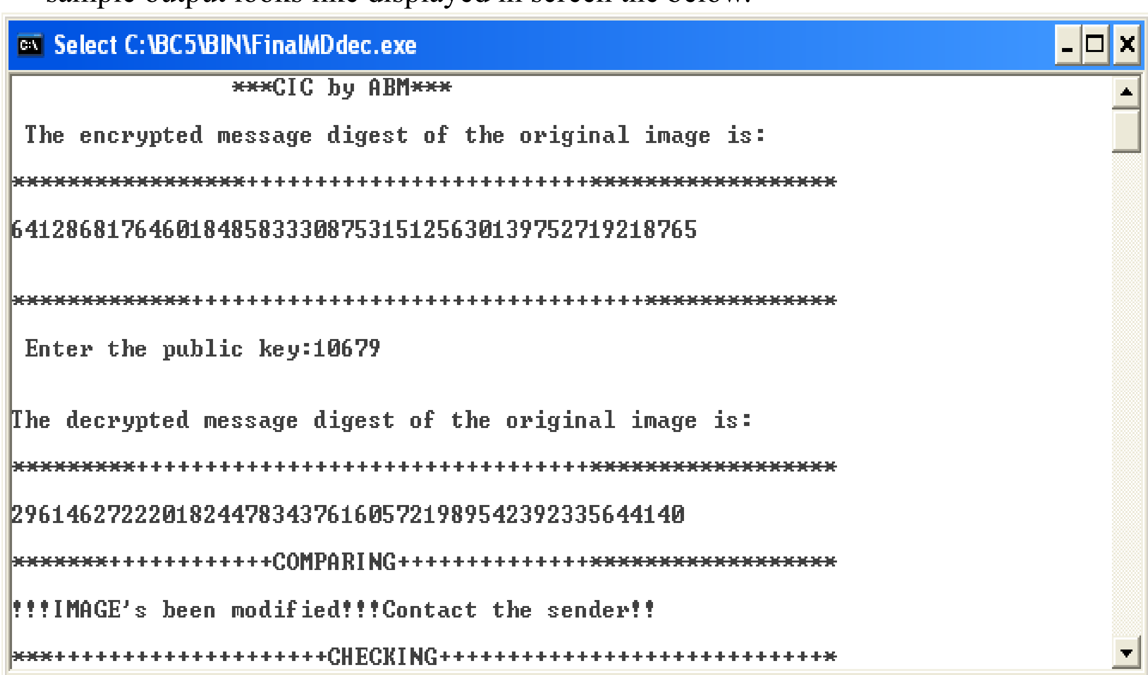


Fig 5.17 Message displayed if the image is hacked during transmission.

Chapter-6
Conclusion and Recommendation

6.1 Conclusion

Image encryption using conventional cryptography is very fast. It doesn't take much time of the processor during execution. It, however, suffers some defects in that its key management (distribution) is not reliable. In this type of encryption process, the same key (a single key) is used both during encryption and decryption processes. In other words, it requires that the sender and the receiver share the same key. As a consequence, key distribution remains to be the sole problem of conventional encryption. That's, the key is very prone to disclosure to unauthorized parties questioning the reliability of the technique. This problem is therefore addressed by employing another technique called asymmetric key encryption.

Asymmetric key encryption, as the name implies, employs two different keys for the encryption and decryption process of the image, which sufficiently alleviates the problem of key distribution. It is implemented using RSA algorithm which makes use of two very large prime numbers to calculate the encryption and decryption keys known as public key and private key, respectively. In this technique of encryption, it is virtually impossible and impractical to figure out what the private key is; the public key is made known and can be factored, though. This is because the prime numbers used are very large. It's however so slow a method that takes much of the processor's time. Further more, it potentially stacks the processor if it is used to encrypt messages comprising a large set of integers like image. It only provides a very high digital security envelope at the expense of the processor speed.

All in all, to optimize the speed and security in the process of the image encryption and decryption, we have exclusively exploited the good points of both techniques. That's, we employed the fast symmetrical DES algorithm for the encryption and decryption of the image. And RSA for the encryption and decryption of the DES key which was selected in a random fashion. What's more, we used the RSA encryption technique for the purpose of authentication and digital signature along with a fast and secure algorithm called Secure Hash Algorithm (SHA-1). The SHA-1 further enhances the security of image communication. It is used to produce message digests by condensing an image of length up to 2^{64} bits irreversibly. The message digest of fixed length 160 bits serves as a digital signature, and verifies authentication.

6.2 Recommendation

We recommend that courts and crime investigation beauraux use this technique for the purpose of exchanging pictures (images) of criminals and alleged people securely thereby making their work and communication very fast. Besides, it could be used for the exchange of medical images that requires high security among different medical hospitals or institutes, and it could be employed for secure video conferencing.

We also recommend that anybody who is interested, may further our work by alleviating the problems we have faced. One major problem of our work is that the message digest which is the function of the image is very sensitive to any kind of change or modification in the image during transmission, even a change of a single bit brings about different message digests. In other words, it is probable that the message digests of the original image and the decrypted image might be different even though the image has not been attacked by hackers during transmission which confuses the sender. This is mainly attributed to the imperfection of the communication channels that introduce bit errors thereby producing different message digests. So, we encourage people to develop methods through which channel errors and attacks can implicitly be identified.

C++ Codes for Image Encryption and Decryption

All of the codes are contained inside the CD in the following case.

References

Books:

Menezes, P. Van Oorschot, S. Vanstone, Hand book of Applied Cryptography, CRC press, 1996

William Stallings, Data and computer communications, 6th edition Prentice Hall, New Jersey, 1996

Bruce Eckel, Thinking in C++, 2nd ed. Volume 1, 2000

Davis Chapman, Teach Yourself Visual C++ in 21 days, SAMS, USA 1998

Digital Image processing

Ashenafi, Daniel, and Tesfamichael, Cryptography for data communication, 2005

Webs:

http://en.wikipedia.org/wiki/Digital_signature

http://orlingrabbe.com/

http://en.wikipedia.org/wiki/SHA_hash_functions

http://en.wikipedia.org/wiki/Message_authentication_code

http://www.koder.com/image