NEAR EAST UNIVERSITY

Faculty of Engineering

Department of Computer Engineering

SECURE COMMUINCATION USING DES

Graduation Project

COM- 400

Student: Mahmoud ALLabadi (20011073)

Supervisor: Prof. Dr. Fakhreddin Mamedov

Nicosia - 2004

ACKNOWLEDGEMENTS

First of all I would like to express sincere gratitude to my project supervisor Professor Dr. Fakhherredin Mamedov for his patient and consistent support. Without his encouragement and direction, this work would not have been completed.

More over I want to pay special regards to my family who are enduring these all expenses and supporting me in all events. I am nothing without their prayers. They also encouraged me in crises. I shall never forget their sacrifices for my education so that I can enjoy my successful life as they are expecting, I will never forget my father, my mother, my brother and sisters. They may get peaceful life in Heaven.

Finally, the best of my acknowledges, I want to honor all my friends who have supported me or helped me in our life. I also pay my special thanks to my all friends who have helped me in my project and gave me their precious time to complete my project, especially Ahmed Zidat, Amjad Humoda, Mahmoud EL-Ali, Magdi AL-Tebe, Mohammed AL-Arrore And Rami Alea.

ABSTRACT

Cryptography protects a message or file from being read by an eavesdropper who has no other means of access to either the original text of what is protected, or the key with which it is encrypted. We can achieve this by using DES algorithm. We use a program to encrypt the text; the program will change the letters into symbols and other weird characters, so when someone opens the file they cannot read it. The interconnection of networks is an increasing trend in government and private industry. There is the obvious danger that connections made in such an extended network may increase the risk of a security compromise, with the owners unaware of the risk. Network connections should therefore be protected, at a level based on the risk.

The aim of this project is to transmit data and preserve its privacy and authentication in critical applications. One of the several data encryption types, Data Encryption Standard (DES) has emerged to be the most commonly used in varying applications.

The selective application of technological and related procedural safeguards is an important responsibility of every Federal organization in providing adequate security to its electronic data systems. The Data Encryption Standard (DES) which may be used by Federal organizations to protect sensitive data. Protection of data during transmission or while in storage may be necessary to maintain the confidentiality and integrity of the information represented by the data. The algorithms uniquely define the mathematical steps required to transform data into a cryptographic cipher and also to transform the cipher back to the original form. The Data Encryption Standard is being made available for use by Federal agencies within the context of a total security program consisting of physical security procedures, good information management practices, and computer system/network access controls.

TABLE OF CONTENTS

INTRODUCTION

Communication and information technology are making a dramatic impact on society and commerce. Digital information can be efficiently stored, processed and communicated, allowing substantial improvements in production and wealth.

Data encryption is used pervasively in today's connected society. The two most basic facets of modern day data encryption are data privacy and authentication. As modern society becomes more connected, and more information becomes available there is a need for safeguards which bring data integrity and data secrecy. In addition, authenticating the source of information gives the recipient, with complete certainty that the information came from the original source and that it has not been altered from its original state. Both, the needs for information privacy and data authentication have motivated cryptography.

The DES cryptographic algorithm converts plaintext to ciphertext using the 56-bit key in the encryption process. The same algorithm is reused with the same key to convert ciphertext back to plaintext, in the decryption process. The algorithm consists of 16 "rounds" of operations that mix the data and key together in a prescribed manner using the fundamental operations of permutation and substitution. The goal is to completely scramble the data and key so that every bit of the ciphertext depends on every bit of the data plus every bit of the key.

The unique key chosen for use in a particular application makes the results of encrypting data using the algorithm unique. Using a different key causes different results. The cryptographic security of the data depends on the security provided for the key used to encrypt and decrypt the data.

My first chapter is all about secure communication and its objectives authentication, data integrity and data privacy.

My second Chapter is all about the introduction as cryptography is the art of limiting the use and access of information to attain secure communication, to address such threats. And what functions involve in this technique and then main encryption and decryption of data.

In my third chapter I have explained various functions techniques used in cryptography in detail. It includes ciphers and the two kinds of cryptography: private key algorithm and public key algorithm and some examples on each of them.

The fourth chapter describes briefly simplified DES (S_DES) and how DES algorithm works in details, and there are a lot of examples which make the understanding of this complex algorithm more easily. And also assigns if is it possible to crack DES algorithm or not.

The final chapter presents implementation S_DES by using C language which it's educational more than secure encryption algorithm.

Conclusion presents the obtained important results and contribution in the project.

1. INTRODUCTION TO SECURE COMMUINCATION

1.1 Overview

When two people wish to communicate over some distance, they will send some form of message. To prevent some enemy from understanding the message, they can encrypt it. If the enemy were to learn the encryption method, he could read the message. It would seem obvious that the method of encryption cannot be transmitted, in the clear, over the communications channel, and still be useful. This, however, is not so. If the two communicants transmit the encryption method in the proper fashion, then they will be able to understand what is going on, but any enemy will become hopelessly confused. Secure communication should provide Privacy, Authentication, and Integrity.

1.2 Secure Communications

The Mutual Authentication procedure ensures that both sender and recipient are authorized to communicate together. This procedure is very convenient when it comes to protecting access to a server and avoiding connection to a wrong address. Smart cards lend themselves excellently to Mutual Authentication. At the beginning of the communication session you check with your smart card whether or not you are connected to the right server. The authentication is managed directly between the smart card and the server or fire-wall and is based on cryptographic algorithms and random numbers. As all the calculations are made internally in the smart card no outsiders have insight into how these computations are performed. Both sender and recipient must be sure that the information communicated over the network has not been modified while being sent. Without security methods an intruder has the capability to modify the data while it is being transferred. Two features based on the same concept can be used to certify the integrity of the information.

The first feature is the digital signature. The digital signature is a set of information calculated from the data to be sent and is therefore unique to each document. The most widely used approach to calculate this signature is based on hashing algorithm. This algorithm reduces the message to a unique smaller set of data. After the exchange, the signature is enciphered using a cryptographic algorithm. Throughout the whole process the digital signature is unique to the sender and to the message itself. The message itself is sent in plain-text with the ciphered signature. If an intruder changes the message, the ciphered signature is no longer correct and the recipient rejects the message.

The second feature is the Cipher text method. This means that the complete message is enciphered by cryptographic algorithm before sending. In this way, no intruder has the capability to understand the message and even less to change it. The recipient receives the Cipher text and uses a decryption algorithm to get the plain-text message.

1.2.1 Secure Communications over Insecure Channels

When two people wish to communicate over some distance, they will send some form of message. When they fear that some enemy, who they do not wish to read the message, might intercept it, then they will encrypt the message. The enemy will then be unable to understand the message, even if he intercepts it, because he does not know how it was encrypted. If the enemy were to learn the encryption method, then he could read the message. Thus, the two people can communicate securely because they have information which is not known to the enemy. This implies that the two people, (call them A and B), have made some form of prior arrangement, while E was unable to listen in. It would thus appear that a necessary precursor to a cryptographically secure communications channel between A and B is the making of prior arrangements or the communication of information over some very special communications channel which is known to be secure already. This is not in fact the case. If A and B have made no prior arrangements, and E can listen to all communications between A and B, they can still establish a cryptographically secure communications channel. The work required of E to break the encryption will increase as the square of the work required of A and B to establish the link. Two agencies, be they computers, people, institutions, or whatever, wish to communicate securely. These two, A and B, have available a communication channel with the following properties:

No message sent by either A or B can be modified by E. E is unable to send false or spurious messages.

E can read every message which is sent. His reception of these messages is as good as that of A and B. He does not occasionally let a message slip by, but receives all of them.

In addition to this, through some massive breach of security by both A and B, E is aware of everything that A and B know. This security breach has just been sealed, and E is no longer able to find out information known to A and B, unless they transmit it on the communications channel.

1.3. Security objectives

The table below explains some of objectives.

Table 1.1 Some information security objectives

Privacy or confidentiality	Keeping information secret from all but those who are authorized to see it.
Data integrity ensuring	Information has not been altered by unauthorized or unknown means.
Entity authentication or identification	Corroboration of the identity of an entity (e.g., a person, a computer terminal, a credit card, etc.).
Message authentication	Corroborating the source of information; also known as data origin authentication.
Signature	A means to bind information to an entity.
Authorization	Conveyance, to another entity, of official sanction to do or be something.
Validation	A means to provide timeliness of authorization to use or manipulate information or resources.
Access control	Restricting access to resources to privileged entities.
Certification	Endorsement of information by a trusted entity.
Time stamping	Recording the time of creation or existence of information.
Witnessing	Verifying the creation or existence of information by an entity other than the creator.
Receipt	Acknowledgement that information has been received.

Confirmation	Acknowledgement that services has been provided.
Ownership	A means to provide an entity with the legal right to use or transfer a resource to others.
Anonymity	Concealing the identity of an entity involved in some process.
Non-repudiation	Preventing the denial of previous commitments or actions.
Revocation	Retraction of certification or authorization.

In later three sections some related concepts belong to security.

1.4 Data Privacy

There are two aspects to determining the level of privacy that can be attained. To begin with, there is an analysis of the security of the two systems from an algorithmic view. The questions rose at this stage aim to consider exactly how hard it is to derive a private or secret key from encrypted text or public keys.

Currently, one of the main secret key algorithms is DES, although two other more recent algorithms, RC2 and RC4 have also arisen. The size (i.e. length) of keys employed in processes is considered to be a useful metric when considering the strength of cryptology. This is because, longer key sizes generally make encrypted text more difficult to decrypt without the appropriate key.

The DES algorithm has a maximum key length of 56 bits. Current consensus is that this range of key size yields keys that are strong enough to withstand attacks using current technologies. The algorithms fixed size nature may, however, constrain it in the future when hardware and theoretic advances are made. The RC2 and RC4 algorithms also have bounded maximum key sizes that limit their usefulness similarly.

A major problem associated with secret key systems, however, is their need for a secure channel within which keys can be propagated. In Kerberos, every client needs to be made aware of its secret key before it can begin communication. To do so without giving away the key to any eavesdroppers requires a secure channel. In practice, maintaining a channel that is completely secure is very difficult and often impractical.

A second aspect to privacy concerns how much inferential information can be obtained through the system. For example, how much information is it possible to deduce without explicitly decrypting actual messages. One particularly disastrous situation would be if it were possible to derive the secret or private keys without mounting attacks on public keys or encrypted messages.

There is a danger that the ability to watch a client progress through the authentication protocol is available. Such information may be enough to mount an attack on the client by jamming the network at strategic points in the protocol. Denial of service like this may be very serious in a time critical system.

In pure algorithmic terms, RSA is a strong. It has the ability to support much longer key lengths than DES etc. Key length is also only limited by technology, and so the algorithm can keep step with increasing technology and become stronger by being able to support longer key lengths.

Unlike secret key systems, the private keys of any public key system need never be transmitted. Provided local security is strong, the overall strength of the algorithm gains from the fact that the private key never leaves the client.

RSA is susceptible to information leakage, however, and some recent theoretic work outlined an attack plan that could infer the private key of a client based on some leaked, incidental information. Overall however, the RSA authentication protocol is not as verbose as the Kerberos equivalent. Having fewer interaction stages limits the bandwidth of any channel though which information may escape. A verbose protocol like Kerberos's simply gives an eavesdropper more opportunity to listen and possibly defines a larger and more identifiable pattern of interaction to listen for.

1.5 Authentication

A system geared primarily towards secure authentication of access requests and identity. It achieves this through a three stage protocol. As clients progress through the protocol they gain more confidence in the server's authenticity based on a protocol whereby a server is deemed trustworthy if it can return a piece of secret information known originally only to the client that is passed as a message to the server. The message is encoded prior to transmission in a key that only the proper destination server can understand. This general algorithm is applied at first to the main repository, which is assumed not to have been compromised; no communication in the system can be trusted until the repository regains integrity.

If a server can understand a message containing some secret piece of information, known only to the originating client initially, that was sent to it in an encrypted form using its own public key, then returning the secret information to the originating client (using its public key) will gain the clients trust. The client may assume that the responding server is legitimate as only the legitimate server could decrypt the original message.

The main sticking point in this protocol believes whether or not the initial message is being encoded using the correct public key. Often to determine the correct public key for a service (if it is not initially known) a client must ask a public key server. An attacker successfully impersonating the public key server may supply the client with a fake key, claiming that it is the correct public key for the required server when, in actuality, the impostor can decrypt the supplied key and is waiting to steal the messages.

RSA uses 'certificates' that can be attached to a reply to authenticate the public key of the sender. The certificates themselves are trusted because they are issued from a higher authority (a Certificate Authority, CA) that, it must be assumed, has validated the contents of the certificate.

The trust of the certificate issuer in this situation is similar to the trust required of the key repository. If can be argued that trust can be broken between the client and certificate issuer. If a false certificate is presented to a trusting client, the client has no defenses and may simply believe the false certificate.

1.6 Data Integrity

RSA, as a public key cryptosystem, supports the notion of digitally signing a document by appending a ``digital signature" to the main body text of the document. To prove that the signature corresponds to the message body, and hasn't been copied from another of the sender's messages by an impostor, each signature is made message specific by the sender before the message is sent. A technique called hashing is used to derive a 'unique' identifier (or "message digest") that corresponds to the message being sent. Each identifier is probabilistically unique to the point that it is unlikely that any other meaningful message may map to the same digest. Well known digest functions MD2, MD4 and MD5 are algorithmically strong in the respect that they produce digests that are probabilistically unique within an appropriately wide context. By encrypting the digest with the private key of the sender, no other person may alter it in transit, except in the unlikely event that they have the private key of the sender. Anyone may decrypt the signature using the sender's public key. This yields the original message digest which can be compared with a hashed version of the received version. If the two digests don't match, then the message has been corrupted or vandalized.

All in all, digital signatures provide an elegant method of detecting unauthorized modifications to information in transit, or even in storage. Performing the hashing operations on top of any standard encryption may incur a cost, but the overall idea is to not have to encrypt bulky general messages in their entirety if they only need protection against modification, rather than against snooping.

The cost of encrypting an entire message would theoretically be larger than the total cost of hashing the entire message into a smaller "digest" and then encrypting that digest. This is only acceptable, however for messages that require protection against modification and not against snooping.

The main limitations of digital signatures are their dependency on an authentic public key. If the receiver is fooled into using the wrong public key then an impostor can craft his own signatures and pass false information. Because all messages are encrypted with the appropriate keys, the transmissions are assured to be secure within the domain
To communicate outside the domain although communication outside the domain is possible, it creates tenuous long links which are possibly more prone to attack. Their size attracts attention and logically there are more points to attack. To be sure of authenticity, all data transmitted needs to be encrypted by the sender using an appropriate key that was gained by communicating.

Communication is an essential part of life. We can say that it marks the progress of human beings. Traditional media for communication are the sending of letters through the Post Office, talking over the phone through the Telecommunications company, or -- more commonly -- to speak directly with the other person. These traditional media have existed for a long period of time and special provisions have been made so that people can communicate in a secure way, either for personal or for business communication. For face–to–face communication, people can recognize each other's physical characteristics or they can compare hand–written signatures with that of official documents like an ID card. Mimicking all of the physical characteristics of a person is difficult. People can accept with a high level of certainty the identity of their colleague. Signature forging is difficult and there are laws that define forging as a crime. The bottom line is that for each communication medium, there is a transitional period when specific laws and technologies are set in order for people to communicate securely and transparently.

The Internet, as a network that interconnects networks of computers around the world, is a new communication medium that is substantially different from existing ones. For example, on the Internet, the communicating parties do not have physical contact. It is rather more difficult for one to disguise oneself to someone else, imitate the voice and other aspects behavior and get information on prior common experiences. On–line transactions do not impose such barriers for illegitimate transactions. Additionally, on the Internet, one can automate the same type of fraud bringing higher gains and a bigger incentive. The law and the technologies to let transparent and secure communication have not been fully defined or set yet.

Since the Cryptography is the science of devising methods that allow information to be secret in a secure form in such a way that the only person able to retrieve this information is the intended recipient, so to attain secure communication cryptography must be applied.

2. INTRODUCTION TO CRYPTOGRAPHY

2.1 Overview

To introduce cryptography, an understanding of issues related to information security in general is necessary. Network security manifests itself in many ways according to the situation and requirement. Regardless of who is involved, to one degree or another, all parties to a transaction must have confidence that certain objectives associated with network security have been met. Often the objectives of on security cannot solely be achieved through mathematical algorithms and protocols alone, but require procedural techniques and abidance of laws to achieve the desired result. One of the fundamental tools used in network security is the signature. It is a building block for many other services such as no repudiation, data origin authentication, identification, and witnessing, to mention a few. Achieving network security in an electronic society requires a vast array of technical and legal skills. There is, however, no guarantee that all of the network security objectives deemed necessary can be adequately met. The technical means is provided through cryptography. Cryptography is not the only means of providing network security, but rather one set of techniques

2.1 Cryptography

Cryptography is the study of mathematical techniques related to aspects of network security such as confidentiality, data integrity, entity authentication, and data origin authentication.

The following are the goals of the Cryptography

1. Confidentiality is a service used to keep the content of information from all but those authorized to have it. There are numerous approaches to providing confidentiality, ranging from physical protection to mathematical algorithms.
2. Data integrity is a service which addresses the unauthorized alteration of data. To assure data integrity, one must have the ability to detect data manipulation by unauthorized parties.

3. Authentication is a service related to identification. This function applies to both entities and information itself. Aspect of cryptography is usually subdivided into two major classes: entity authentication and data origin authentication.
4. Non-repudiation is a service which prevents an entity from denying previous commitments or actions.

A fundamental goal of cryptography is to adequately address these four areas in both theory and practice. Cryptography is about the prevention and detection of cheating and other malicious activities. A number of basic cryptographic tools (primitives) used to provide network security. Examples of primitives include encryption schemes hash functions, and digital signature schemes. Figure 2.1 provides a schematic listing of the primitives considered and how they relate.

These primitives should be evaluated with respect to various criteria such as:

1. Level of security. This is usually difficult to quantify. Often it is given in terms of the number of operations required to defeat the intended objective.
2. Functionality. Primitives will need to be combined to meet various network security objectives. Which primitives are most effective for a given objective will be determined by the basic properties of the primitives.

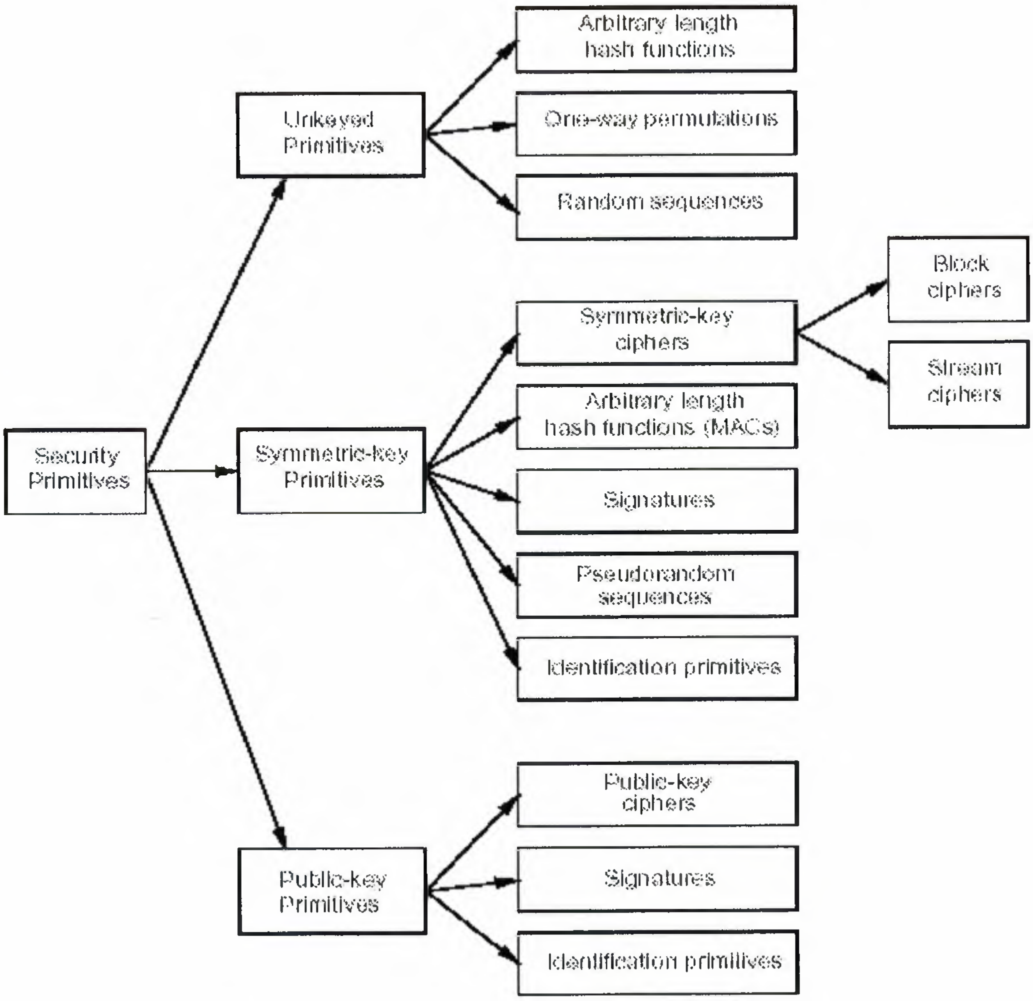


Figure 2.1 A taxonomy of cryptographic primitives.

3. Methods of operation. Primitives, when applied in various ways and with various inputs, will typically exhibit different characteristics; thus, one primitive could provide very different functionality depending on its mode of operation or usage.
4. Performance. This refers to the efficiency of a primitive in a particular mode of operation.
5. Ease of implementation. This refers to the difficulty of realizing the primitive in a practical instantiation. This might include the complexity of implementing the primitive in either a software or hardware environment. The relative importance of various criteria is very much dependent on the application and resources available. For example, in an environment where computing power is limited one may have

to trade off a very high level of security for better performance of the system as a whole.

2.3 Basic Functions and Concepts

A familiarity with basic mathematical concepts used in cryptography will be useful. One concept which is absolutely fundamental to cryptography is that of a function in the mathematical sense. A function is alternately referred to as a mapping or a transformation.

2.3.1 Function

A set consists of distinct objects which are called elements of the set. For example, a set X might consist of the elements a, b, c, and this is denoted X = {a; b; c}. If x is an element of X (usually written x∈X) the image of x is the element in Y which the rule *f* associates with x; the image y of x is denoted by y = *f*(x). Standard notation for a function *f* from set X to set Y is f: X→Y.

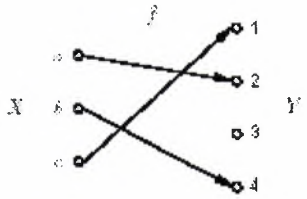


Figure 2.2 A function *f* from a set X to a set Y.

- 1-1 Functions: A function is 1 - 1 (one-to-one) if each element in the co domain Y is the image of at most one element in the domain X.
- Onto function: A function is onto if each element in the co domain Y is the image of at least one element in the domain.
- Bijection: If a function f: X→Y is 1-1 and Im (*f*) = Y, then *f* is called a bijection.
- One-way functions: A function *f* from a set X to a set Y is called a one-way function if *f* (x) is easy to compute for all x∈X but for essentially all elements y∈Im (f) it is "computationally infeasible" to find any x∈X such that *f*(x) = y.
- Trapdoor one-way functions: A trapdoor one-way function is a one-way function *f*: X→Y with the additional property that given some extra

- Permutations: Let S be a finite set of elements. A permutation p on S is a bijection from S to itself (i.e., p: S→S).
- Involutions: Involutions have the property that they are their own inverses. (i.e., *f*: S→ S).

2.3.2 Basic Terminology and Concepts

The scientific study of any discipline must be built upon exact definitions arising from fundamental concepts. Where appropriate, strictness has been sacrificed for the sake of clarity.

2.3.2.1. Encryption Domains and Co-domains

- A denotes a finite set called the alphabet of definition.
- M denotes a set called the message space. M consists of strings of symbols from an alphabet. An element of M is called a plaintext message or simply a plaintext.
- C denotes a set called the ciphertext space. C consists of strings of symbols from an alphabet; differ from the alphabet of M. An element of C is called a ciphertext.

2.3.2.2 Encryption and Decryption Transformations

- K denotes a set called the key space. An element of K is called a key.
- Each element $e \in K$ uniquely determines a bijection from M to C, denoted by Ee.
- D_d denotes a bijection from C to M and D_d is called a decryption function.
- The process of applying the transformation Ee to a message $m \in M$ is usually referred to as encrypting m or the encryption of m.
- The process of applying the transformation D_d to a cipher text c is usually referred to as decrypting c or the decryption of c.
- The keys e and d are referred to as a key pair and denoted by (e; d).

2.3.2.3 Achieving Confidentiality

An encryption scheme may be used as follows for the purpose of achieving confidentiality. Two parties Alice and Bob first secretly choose or secretly exchange a key pair (e; d). At a subsequent point in time, if Alice wishes to send a message $m \in M$ to Bob, she computes c = Ee (m) and transmits this to Bob. Upon receiving c, Bob computes D_d (c) = m and hence recovers the original message m.

The question arises as to why keys are necessary. If some particular encryption/decryption transformation is exposed then one does not have to redesign the entire scheme but simply change the key. Figure 2.3 provides a simple model of a two-party communication using encryption.

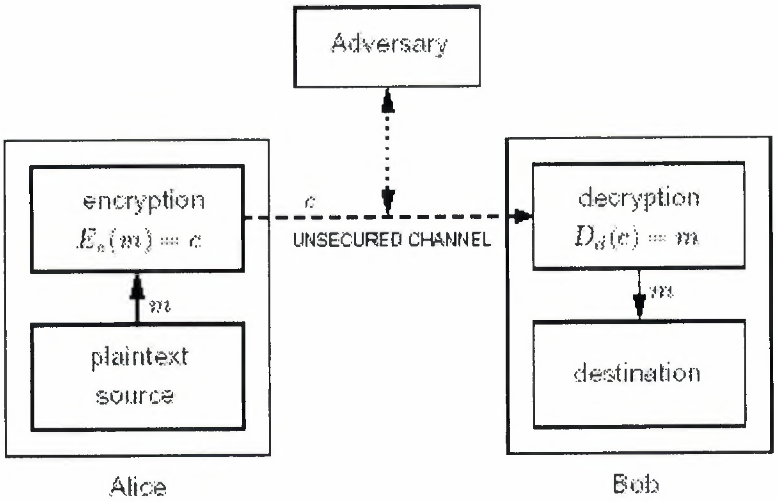


Figure 2.3 Schematic of a two-party communication.

2.3.2.4 Communication Participants

Referring to Figure 1.3, the following terminology is defined.

- An entity or party is someone or something which sends, receives, or manipulates information. An entity may be a person, a computer terminal, etc.
- A sender is an entity in a two-party communication which is the legitimate transmitter of information.
- A receiver is an entity in a two-party communication which is the intended recipient of information.
- An adversary is an entity in a two-party communication which is neither the sender nor receiver, and which tries to defeat the information security service being provided between the sender and receiver.

2.3.2.5. Channels

A channel is a means of conveying information from one entity to another. A physically secure channel is one which is not physically accessible to the adversary. An unsecured channel is one from which parties other than those for which the information is intended can reorder, delete, insert, or read. A secured channel is one from which an adversary does not have the ability to reorder, delete, insert, or read. A secured channel may be secured by physical or cryptographic techniques.

2.3.2.6 Security

A fundamental principle in cryptography is that the sets M; C; K; {Ee: e $\in$ K}, {D_d: d$\in$K} are public knowledge. When two parties wish to communicate securely using an encryption scheme, the only thing that they keep secret is the particular key pair (e; d), which they must select. One can gain additional security by keeping the class of encryption and decryption transformations secret but one should not base the security of the entire scheme on this approach. An encryption scheme is said to be breakable if a third party, without prior knowledge of the key pair (e; d) can systematically recover plaintext from corresponding ciphertext within some appropriate time frame. An encryption scheme can be broken by trying all possible keys to see which one the communicating parties are using. This is called an exhaustive search of the key space.

Frequently cited in the literature are Kerckhoffs' desiderata, a set of requirements for cipher systems. They are given here essentially as Kerckhoffs originally stated them:

1. The system should be, if not theoretically unbreakable, unbreakable in practice.
2. Compromise of the system details should not inconvenience the correspondents.
3. The key should be remember able without notes and easily changed.
4. The cryptogram should be transmissible by telegraph.
5. The encryption apparatus should be portable and operable by a single person.
6. The system should be easy, requiring neither the knowledge of a long list of rules nor mental strain.

2.3.2.7 Network Security in General

So far the terminology has been restricted to encryption and decryption with the goal of privacy in mind. Network security is much broader, encompassing such things as authentication and data integrity.

- A network security service is a method to provide specific aspect of security.
- Breaking a network security service implies defeating the objective of the intended service.
- A passive adversary is an adversary who is capable only of reading information from an unsecured channel.
- An active adversary is an adversary who may also transmit, alter, or delete information on an unsecured channel.

2.4 Symmetric-key Encryption

Consider an encryption scheme consisting of the sets of encryption and decryption transformations {Ee: $e \in K$} and {D_d : $d \in K$}, respectively, where K is the key space. The encryption scheme is said to be symmetric-key if for each associated encryption/decryption key pair (e; d), it is computationally easy to determine d knowing only e, and to determine e from d. Since e = d in most practical symmetric-key encryption schemes, the term symmetric key becomes appropriate.

A two-party communication using symmetric-key encryption can be described by the block diagram of Figure 2.4, with the addition of the secure channel.

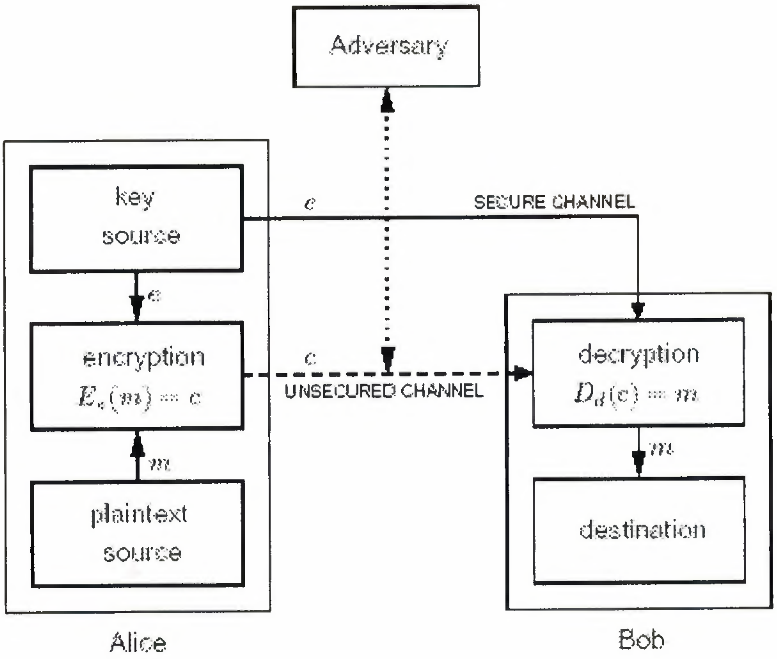


Figure 2.4 Two-party communication using encryption, with a secure channel

One of the major issues with symmetric-key systems is to find an efficient method to agree upon and exchange keys securely. It is assumed that all parties know the set of encryption/decryption transformations there are two classes of symmetric-key encryption schemes which are commonly distinguished, block ciphers and stream ciphers.

2.4.1 Block Ciphers

A block cipher is an encryption scheme which breaks up the plaintext messages to be transmitted into strings (called blocks) of a fixed length t over an alphabet A, and encrypts one block at a time. Most well-known symmetric-key encryption techniques are block ciphers. Two important classes of block ciphers are substitution ciphers and transposition ciphers

2.4.2 Stream Ciphers

Stream ciphers form an important class of symmetric-key encryption schemes. They are, in one sense, very simple block ciphers having block length equal to one. What makes them useful is the fact that the encryption transformation can change for each symbol of plaintext being encrypted. In situations where transmission errors are highly probable, stream ciphers are advantageous because they have no error propagation. They can also be used when the data must be processed one symbol at a time

2.4.3 The Key Space

The size of the key space is the number of encryption/decryption key pairs that are available in the cipher system. A key is typically a compact way to specify the encryption transformation to be used. For example, a transposition cipher of block length t has t! Encryption functions from which to select. Each can be simply described by a permutation which is called the key.

2.5 Digital Signatures

A cryptographic primitive who is fundamental in authentication, authorization, and non-repudiation is the digital signature. The purpose of a digital signature is to provide a means for an entity to bind its identity to a piece of information. The process of signing entails transforming the message and some secret information held by the entity into a tag called a signature.

2.5.1. Nomenclature and Set-up

The transformations S_A and V_A provide a digital signature scheme for A.

- M is the set of messages which can be signed.
- S is a set of elements called signatures, possibly binary strings of a fixed length.
- S_A is a transformation from the message set M to the signature set S, and is called a signing transformation for entity A.
- V_A is a transformation from the set M x S to the set {true, false} V_A is called a verification transformation for A's signatures, is publicly known, and is used by other entities to verify signatures created by A.

2.6 Public-key Cryptography

The concept of public-key encryption is simple and elegant, but has far-reaching consequences. Let {Ee: e ∈ K} be a set of encryption transformations, and let {D_d: d ∈ K} be the set of corresponding decryption transformations, where K is the key space. Consider any pair of associated encryption/decryption transformations (Ee; Dd) and suppose that each pair has the property that knowing Ee it is computationally infeasible, given a random ciphertext c∈C, to find the message m∈M such that Ee(m) = c. This property implies that given e it is infeasible to determine the corresponding decryption key d. Ee is being viewed here as a trapdoor one-way function with d being the trapdoor information necessary to compute the inverse function and hence allow decryption. This is unlike symmetric-key ciphers where e and d are essentially the same.

The encryption method is said to be a public-key encryption scheme if for each associated encryption/decryption pair (e; d), one key e (the public key) is made publicly available, while the other d (the private key) is kept secret. For the scheme to be secure, it must be computationally infeasible to compute d from e. To avoid ambiguity, a common convention is to use the term private key in association with public-key cryptosystems, and secret key in association with symmetric-key cryptosystems

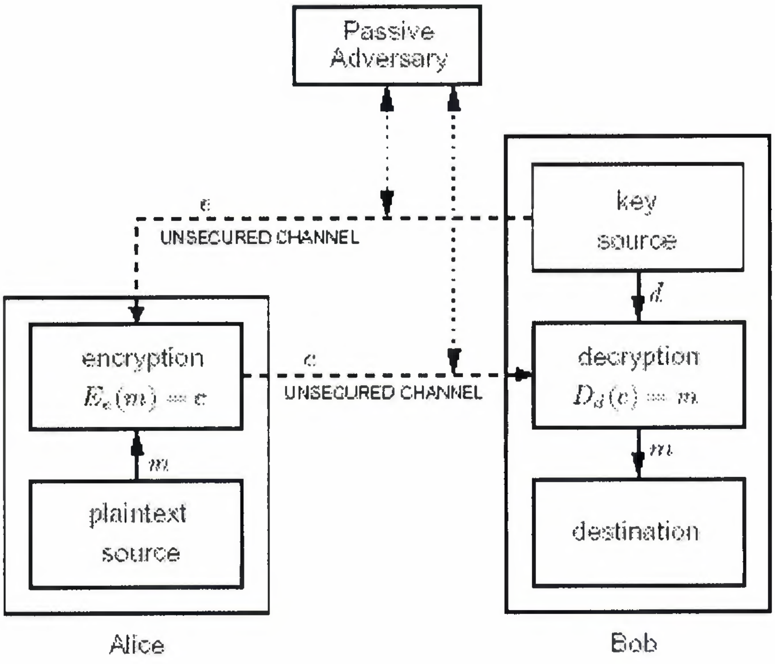


Figure 2.5 Encryption using public-key techniques.

2.7 Hash Functions

One of the fundamental primitives in modern cryptography is the cryptographic hash function, often informally called a one-way hash function. A simplified definition for the present discussion follows. A hash function is a computationally efficient function mapping binary strings of arbitrary length to binary strings of some fixed length, called hash-values. For a hash function which outputs n-bit hash-values and has desirable properties, the probability that a randomly chosen string gets mapped to a particular n-bit hash-value (image) is 2^{-n}. The basic idea is that a hash-value serves as a compact representative of an input string. To be of cryptographic use, a hash function h is typically chosen such that it is computationally infeasible to find two distinct inputs which hash to a common value and that given a specific hash-value y, it is computationally infeasible to find an input x such that h(x) = y. The most common cryptographic uses of hash functions are with digital signatures and for data integrity Hash functions are typically publicly known and involve no secret keys. When used to detect whether the message input has been altered, they are called modification detection codes (MDCs). Related to these are hash functions which involve a secret key, and provide data origin authentication as well as data integrity; these are called message authentication codes (MACs).

2.8 Protocols, Mechanisms

A cryptographic protocol is a distributed algorithm defined by a sequence of steps precisely specifying the actions required of two or more entities to achieve a specific security objective. As opposed to a protocol, a mechanism is a more general term encompassing protocols, algorithms and non-cryptographic techniques to achieve specific security objectives. Protocols play a major role in cryptography and are essential in meeting cryptographic goals. Encryption schemes, digital signatures, hash functions, and random number generation are among the primitives which may be utilized to build a protocol.

2.8.1 Protocol and Mechanism Failure

A protocol failure or mechanism failure occurs when a mechanism fails to meet the goals for which it was intended. Protocols and mechanisms may fail for a number of reasons:

1. Weaknesses in a particular cryptographic primitive which may be amplified by the protocol or mechanism.
2. Claimed or assumed security guarantees which are overstated or not clearly understood.
3. The oversight of some principle applicable to a broad class of primitives such as encryption.

When designing cryptographic protocols and mechanisms, the following two steps are essential:

1. Identify all assumptions in the protocol or mechanism design.
2. For each assumption, determine the effect on the security objective if that assumption is violated.

2.9 Classes of Attacks and Security Models

Over the years, many different types of attacks on cryptographic primitives and protocols have been identified. The attacks these adversaries can mount may be classified as follows:

1. A passive attack is one where the adversary only monitors the communication channel. A passive attacker only threatens confidentiality of data.
2. An active attack is one where the adversary attempts to delete, add, or in some other way alter the transmission on the channel.

A passive attack can be further subdivided into more specialized attacks for deducing plaintext from ciphertext.

2.9.1 Attacks on Encryption Schemes

The objective of the following attacks is to systematically recover plaintext from ciphertext, or even more drastically, to deduce the decryption key.

1. A ciphertext-only attack is one where the adversary tries to deduce the decryption key or plaintext by only observing ciphertext.
2. A known-plaintext attack is one where the adversary has a quantity of plaintext and corresponding ciphertext.

3. A chosen-plaintext attack is one where the adversary chooses plaintext and is then given corresponding ciphertext.
4. An adaptive chosen-plaintext attack is a chosen-plaintext attack wherein the choice of plaintext may depend on the ciphertext received from previous requests.
5. A chosen-ciphertext attack is one where the adversary selects the ciphertext and is then given the corresponding plaintext. One way to mount such an attack is for the adversary to gain access to the equipment used for decryption
6. An adaptive chosen-ciphertext attack is a chosen-ciphertext attack where the choice of ciphertext may depend on the plaintext received from previous requests.

2.9.2 Attacks on Protocols

The following is a partial list of attacks which might be mounted on various protocols. Until a protocol is proven to provide the service intended, the list of possible attacks can never be said to be complete.

1. Known-key attack. In this attack an adversary obtains some keys used previously and then uses this information to determine new keys.
2. Replay. In this attack an adversary records a communication session and replays the entire session, or a portion thereof, at some later point in time.
3. Impersonation. Here an adversary assumes the identity of one of the legitimate parties in a network.
4. Dictionary. This is usually an attack against passwords. An adversary can take a list of probable passwords; hash all entries in this list, and then compare this to the list of true encrypted passwords with the hope of finding matches.
5. Forward search. This attack is similar in spirit to the dictionary attack and is used to decrypt messages.
6. Interleaving attack. This type of attack usually involves some form of impersonation in an authentication protocol.

3. CRYPTOGRAPHY FUNCTIONS

3.1 Overview

In this chapter basic functions involved in cryptography are explained. Functions which are used in the encryptions and decryption of the text such ciphers mainly block cipher and. Hash functions are also one of the important encryption functions. It is also explained that how the attacks are being done on cryptography and what are the authentication methods are being used so for.

3.2 Block Cipher

The most important symmetric algorithms are block ciphers. The general operation of all block ciphers is the same - a given number of bits of plaintext (a block) are encrypted into a block of ciphertext of the same size. Thus, all block ciphers have a natural block size, the number of bits they encrypt in a single operation. This stands in contrast to stream ciphers, which encrypt one bit at a time. Any block cipher can be operated in one of several modes.

3.2.1 Iterated Block Cipher

An iterated block cipher is one that encrypts a plaintext block by a process that has several rounds. In each round, the same transformation or round function is applied to the data using a sub key. The set of sub keys are usually derived from the user-provided secret key by a key schedule. The number of rounds in an iterated cipher depends on the desired security level and the consequent trade-off with performance. In most cases, an increased number of rounds will improve the security offered by a block cipher, but for some ciphers the number of rounds required to achieve adequate security will be too large for the cipher to be practical or desirable.

3.2.2 Electronic Codebook (ECB) Mode

ECB is the simplest mode of operation for a block cipher. The input data is padded out to a multiple of the block size, broken into an integer number of blocks, each of which is encrypted independently using the key. In addition to simplicity, ECB has the advantage of allowing any block to be decrypted independently of the others. Thus, lost data blocks do not affect the decryption of other blocks. The disadvantage of ECB is that it aids known-plaintext attacks. If the same block of plaintext is encrypted twice with ECB, the two resulting blocks of cipher text will be the same.

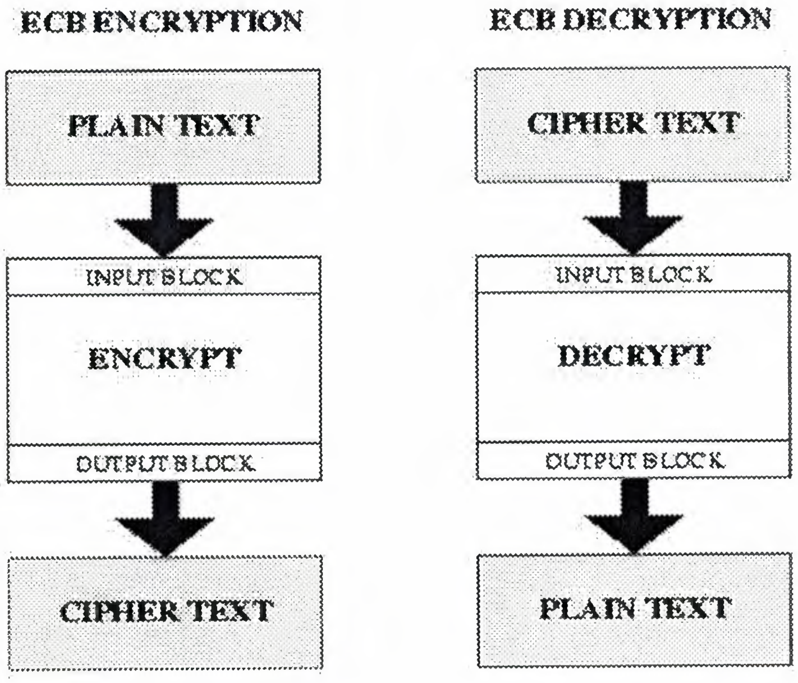


Figure 3.1 Shows a ECB Encryption/Decryption Model

3.2.3 Cipher Block Chaining (CBC) Mode

CBC is the most commonly used mode of operation for a block cipher. Prior to encryption, each block of plaintext is XOR-ed with the prior block of ciphertext. After decryption, the output of the cipher must then be XOR-ed with the previous ciphertext to recover the original plaintext. The first block of plaintext is XOR-ed with an initialization vector (IV), which is usually a block of random bits transmitted in the clear. CBC is more secure than ECB because it effectively scrambles the plaintext prior to each encryption step. Since the ciphertext is constantly changing, two identical blocks of plaintext will encrypt to two different blocks of ciphertext. The disadvantage of CBC is that the encryption of a data block becomes dependent on all the blocks prior to it. A lost block of data will also prevent decoding of the next block of data. CBC can be used to convert a block cipher into a hash algorithm. To do this, CBC is run repeatedly on the input data, and all the ciphertext is discarded except for the last block, which will depend on all the data blocks in the message. This last block becomes the output of the hash function.

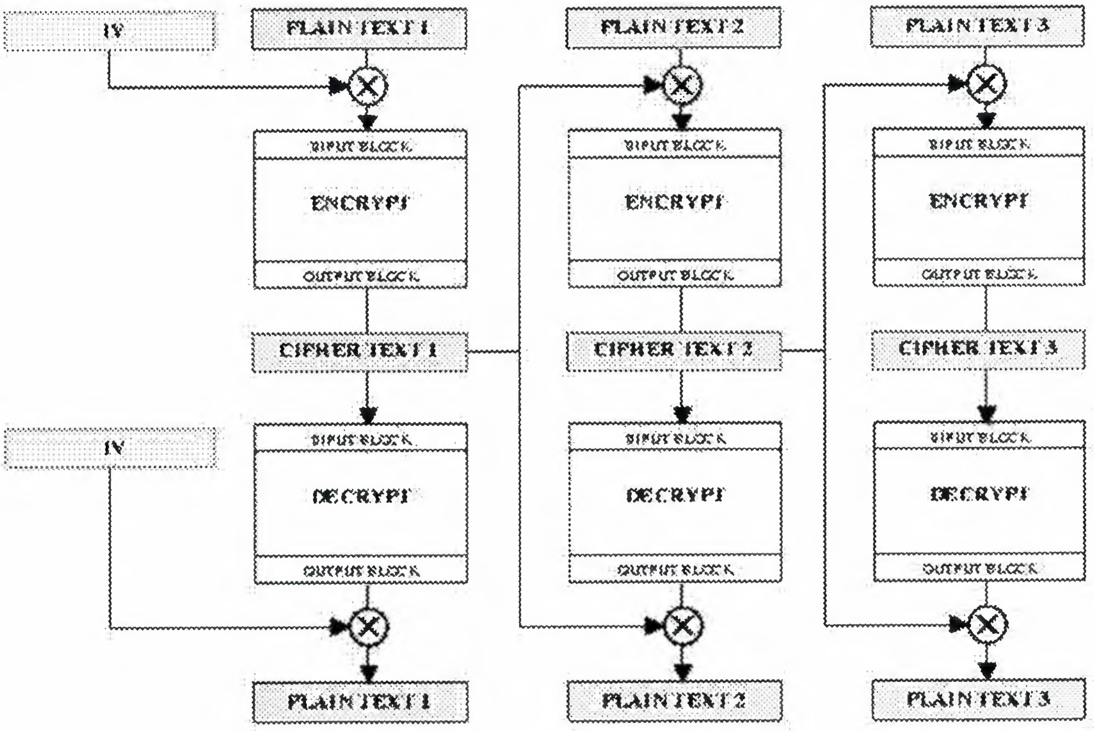


Figure 3.2 Shows a CBC Encryption/Decryption Model

3.2.4 Feistel Ciphers

The figure shows the general design of a Feistel cipher, a scheme used by almost all modern block ciphers. The input is broken into two equal size blocks, generally called left (L) and right (R), which are then repeatedly cycled through the algorithm. At each cycle, a hash function (f) is applied to the right block and the key, and the result of the hash is XOR-ed into the left block. The blocks are then swapped. The XOR-ed result becomes the new right block and the unaltered right block becomes the left block. The process is then repeated a number of times.

The hash function is just a bit scrambler. The correct operation of the algorithm is not based on any property of the hash function, other than it is completely deterministic; i.e. if it's run again with the exact same inputs, identical output will be produced. To decrypt, the ciphertext is broken into L and R blocks, and the key and the R block are run through the hash function to get the same hash result used in the last cycle of encryption; notice that the R block was unchanged in the last encryption cycle. The hash is then XOR'ed into the L block to reverse the last encryption cycle, and the process is repeated until all the encryption cycles have been backed out. The security of a Feistel cipher depends primarily on the key size and the irreversibility of the hash function. The output of the hash function should appear to be random bits from which nothing can be determined about the inputs.

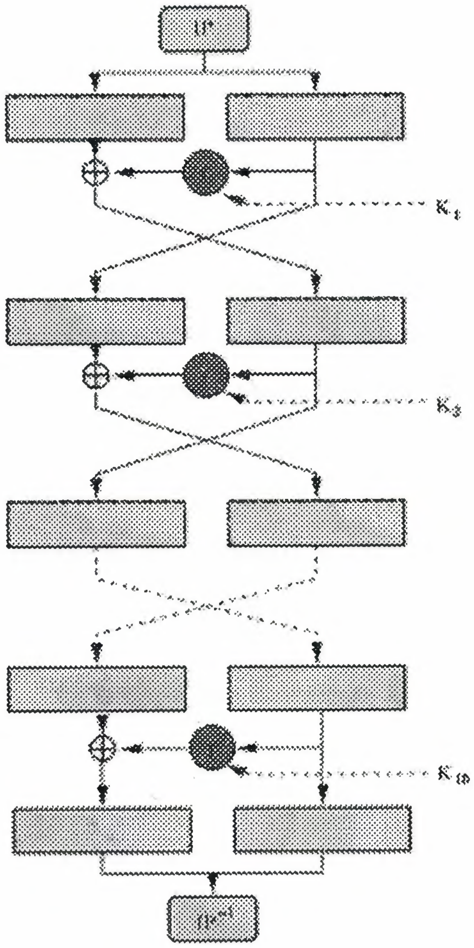


Figure 3.3: Shows a Feistel Model

3.3 Authentication Confirms an Identity

Authentication is the process of confirming an identity. In the context of network interactions, authentication involves the confident identification of one party by another party. Authentication over networks can take many forms. Certificates are one way of supporting authentication.

Network interactions typically take place between a client, such as browser software running on a personal computer, and a server, such as the software and hardware used to host a Web site. Client authentication refers to the confident identification of a client by a server (that is, identification of the person assumed to be using the client software). Server authentication refers to the confident identification of a server by a client (that is, identification of the organization assumed to be responsible for the server at a particular network address).

Client and server authentication are not the only forms of authentication that certificates support. For example, the digital signature on an email message, combined with the certificate that identifies the sender, provide strong evidence that the person identified by that certificate did indeed send that message. Similarly, a digital signature on an HTML form, combined with a certificate that identifies the signer, can provide evidence, after the fact, that the person identified by that certificate did agree to the contents of the form. In addition to authentication, the digital signature in both cases ensures a degree of non-repudiation--that is, a digital signature makes it difficult for the signer to claim later not to have sent the email or the form.

Client authentication is an essential element of network security within most intranets or extranets. The sections that follow contrast two forms of client authentication:

- Password-Based Authentication. Almost all server software permits client authentication by means of a name and password. For example, a server might require a user to type a name and password before granting access to the server. The server maintains a list of names and passwords; if a particular name is on the list, and if the user types the correct password, the server grants access.

- Certificate-Based Authentication. Client authentication based on certificates is part of the SSL protocol. The client digitally signs a randomly generated piece of data and sends both the certificate and the signed data across the network. The server uses techniques of public-key cryptography to validate the signature and confirm the validity of the certificate.

3.4 Symmetric-Key Algorithms

With symmetric-key encryption, the encryption key can be calculated from the decryption key and vice versa. With most symmetric algorithms, the same key is used for both encryption and decryption.

Implementations of symmetric-key encryption can be highly efficient, so that users do not experience any significant time delay as a result of the encryption and decryption. Symmetric-key encryption also provides a degree of authentication, since information encrypted with one symmetric key cannot be decrypted with any other symmetric key. Thus, as long as the symmetric key is kept secret by the two parties using it to encrypt communications, each party can be sure that it is communicating with the other as long as the decrypted messages continue to make sense.

Symmetric-key encryption is effective only if the symmetric key is kept secret by the two parties involved. If anyone else discovers the key, it affects both confidentiality and authentication. A person with an unauthorized symmetric key not only can decrypt messages sent with that key, but can encrypt new messages and send them as if they came from one of the two parties who were originally using the key.

Symmetric-key encryption plays an important role in the SSL protocol, which is widely used for authentication, tamper detection, and encryption over TCP/IP networks. SSL also uses techniques of public-key encryption, which is described in the next section.

3.4.1 Data Encryption Standard (DES)

DES is a Feistel-type Substitution-Permutation Network (SPN) cipher. DES uses a 56-bit key which can be broken using brute-force methods, and is now considered obsolete. A 16 cycle Feistel system is used, with an overall 56-bit key permuted into 16 48-bit sub keys, one for each cycle. To decrypt, the identical algorithm is used, but the order of sub keys is reversed. The L and R blocks are 32 bits each, yielding an overall block size of 64 bits. The hash function "f", specified by the standard using the so-called "S-boxes", takes a

32-bit data block and one of the 48-bit sub keys as input and produces 32 bits of output. Sometimes DES is said to use a 64-bit key, but 8 of the 64 bits are used only for parity checking, so the effective key size is 56 bits, (you can see DES algorithm in details in next chapter).

3.4.2 Triple DES

Triple DES was developed to address the obvious flaws in DES without designing a whole new cryptosystem. Triple DES simply extends the key size of DES by applying the algorithm three times in succession with three different keys. The combined key size is thus 168 bits (3 times 56), beyond the reach of brute-force techniques such as those used by the EFF DES Cracker. Triple DES has always been regarded with some suspicion, since the original algorithm was never designed to be used in this way, but no serious flaws have been uncovered in its design, and it is today a viable cryptosystem used in a number of Internet protocols.

3.5 Asymmetric key Algorithms

The most commonly used implementations of public-key encryption are based on algorithms patented by RSA Data Security. Therefore, this section describes the RSA approach to public-key encryption.

Public-key encryption (also called asymmetric encryption) involves a pair of keys--a public key and a private key--associated with an entity that needs to authenticate its identity electronically or to sign or encrypt data. Each public key is published, and the corresponding private key is kept secret. Data encrypted with your public key can be decrypted only with your private key

When you are using public key, only you will be able to read data encrypted using this key. In general, to send encrypted data to someone, you encrypt the data with that person's public key, and the person receiving the encrypted data decrypts it with the corresponding private key.

Compared with symmetric-key encryption, public-key encryption requires more computation and is therefore not always appropriate for large amounts of data. However, it's possible to use public-key encryption to send a symmetric key, which can then be used to encrypt additional data. This is the approach used by the SSL protocol.

As it happens, the reverse of public key also works: data encrypted with your private key can be decrypted only with your public key. This would not be a desirable way to encrypt sensitive data, however, because it means that anyone with your public key, which is by definition published, could decrypt the data. Nevertheless, private-key encryption is useful, because it means you can use your private key to sign data with your digital signature--an important requirement for electronic commerce and other commercial applications of cryptography. Client software such as Communicator can then use your public key to confirm that the message was signed with your private key and that it hasn't been tampered with since being signed. Digital Signatures and subsequent sections describe how this confirmation process works.

3.5.1 RSA

RSA stands for the initials of the three men Ron Rivest, Adi Shamir, and Len Adleman. The security behind RSA lies in the difficulty of factoring large numbers into their primes. The process involves selecting two large (hundreds of digits) prime numbers (p and q), and multiplying them together to get the sum, n. These numbers are passed through a mathematical algorithm to determine the public key KU = {e, n} and the private key KR = {d, n}, which are mathematically related (the necessary equations are given at the bottom of the page). It is extremely difficult to determine e and/or d given n, thus the security of the algorithm. Once the keys have been created a message can be encrypted in blocks, and passed though the following equation:

$C = M^e \bmod n$

Where C is the ciphertext, M is the plaintext, and e is the recipient's public key. Similarly, the above message could be decrypted by the following equation:

$M = C^d \bmod n$

Where d is the recipient's private key. For example: let's assume that our M is 19 (we will use smaller numbers for simplicity, normally theses numbers would be much larger). We will use 7 as p and 17 as q. Thus, n = 7 * 17 = 119. Our e is then calculated to be 5 and d is calculated to be 77. Thus our KU is {5, 119} and our KR is {77, 119}. We can then pass the needed values through equation (1) to compute C. In this case C is 66. We could then decrypt C (66) to get back our original plain text. We pass the needed values through equation (2) and get 19, our original plaintext! Try it yourself with other numbers.

Note: To determine e and d, perform the following:

Calculate f (n) = (p - 1) (q - 1)

Choose e to be relatively prime to f (n) and less than f(n).

Determine d such that de = 1 mod f (n) and d < f (n).

3.5.2Diffie and Hellman's Contribution

The problem with symmetric keys is that because they can be used both to encrypt and to decrypt, they must be kept very secret. Before any messages are sent, the sender and the receiver must communicate the key very secretly. If the key is found by anyone, they can use it to snoop on the messages. But this limitation is a severe one. If I want to send sensitive information to someone I've never met, perhaps my credit card number to purchase an item, must I first meet with him to set up a secure key? Clearly this is not ideal.

Diffie and Hellman solved this problem by devising a coding scheme called public key cryptography. Actually there are two keys, one public the other private. The public key is used for encoding messages and the private one for decrypting them. It's like a strong box which uses one key to lock up the information and another key to open it.

If I wish to use such a system, I can generate my two keys and give everyone my public key for them to use to encrypt messages they wish to send to me. Only I can decrypt them with my private key. Any one, who wishes to receive encoded messages from me, can do likewise. That is they can generate two keys and send me their public key for encoding messages to them.

3.6 Hash Functions

Hash Functions take a block of data as input, and produce a hash or message digest as output. The usual intent is that the hash can act as a signature for the original data, without revealing its contents. Therefore, it's important that the hash function be irreversible - not only should it be nearly impossible to retrieve the original data, it must also be unfeasible to construct a data block that matches some given hash value. Randomness, however, has no place in a hash function, which should completely deterministic. Given the exact same input twice, the hash function should always produce the same output. Even a single bit changed in the input, though, should produce a different hash value. The hash value should be small enough to be manageable in further manipulations, yet large enough to prevent an attacker from randomly finding a block of data that produces the same hash.

MD5, documented in RFC 1321, is perhaps the most widely used hash function at this time. It takes an arbitrarily sized block of data as input and produces a 128-bit (16-byte) hash. It uses bitwise operations, addition, and a table of values based on the sine function to process the data in 64-byte blocks. RFC 1810 discusses the performance of MD5, and presents some speed measurements for various architectures.

Hash functions can't be used directly for encryption, but are very useful for authentication. One of the simplest uses of a hash function is to protect passwords. UNIX systems, in particular, will apply a hash function to a user's password and store the hash value, not the password itself. To authenticate the user, a password is requested, and the response runs through the hash function. If the resulting hash value is the same as the one stored, then the user must have supplied the correct password, and is authenticated. Since the hash function is irreversible, obtaining the hash values doesn't reveal the passwords to an attacker. In practice, though, people will often use guessable passwords, so obtaining the hashes might reveal passwords to an attacker who, for example, hashes all the words in the dictionary and compares the results to the password hashes.

Another use of hash functions is for interactive authentication over the network. Transmitting a hash instead of an actual password has the advantage of not revealing the password to anyone sniffing on the network traffic. If the password is combined with some changing value, then the hashes will be different every time, preventing an attacker from using an old hash to authenticate again. The server sends a random challenge to the client, which combines the challenge with the password, computes the hash value, and sends it back to the server. The server, possessing both the stored secret password and the random challenge, performs the same hash computation, and checks its result against the reply from the client. If they match, then the client must know the password to have correctly computed the hash value. Since the next authentication would involve a different random challenge, the expected hash value would be different, preventing an attacker from using a replay attack. Thus, hash functions, though not encryption algorithms in their own right, can be used to provide significant security services, mainly identity authentication.

3.7 Digital Signatures

Encryption and decryption address the problem of eavesdropping, one of the three Internet security. But encryption and decryption, by themselves, do not address the other two problems mentioned in Internet Security Issues: tampering and impersonation.

This section describes how public-key cryptography addresses the problem of tampering. Tamper detection and related authentication techniques rely on a mathematical function called a one-way hash (also called a message digest). A one-way hash is a number of fixed lengths with the following characteristics:

- The value of the hash is unique for the hashed data. Any change in the data, even deleting or altering a single character, results in a different value.
- The content of the hashed data cannot, for all practical purposes, be deduced from the hash--which is why it is called "one-way."

As mentioned in Public-Key Encryption, it's possible to use your private key for encryption and your public key for decryption. Although this is not desirable when you are encrypting sensitive information, it is a crucial part of digitally signing any data. Instead of encrypting the data itself, the signing software creates a one-way hash of the data, and then uses your private key to encrypt the hash. The encrypted hash, along with other information, such as the hashing algorithm, is known as a digital signature.

Two items transferred to the recipient of some signed data: the original data and the digital signature, which is basically a one-way hash (of the original data) that has been encrypted with the signer's private key. To validate the integrity of the data, the receiving software first uses the signer's public key to decrypt the hash. It then uses the same hashing algorithm that generated the original hash to generate a new one-way hash of the same data.

(Information about the hashing algorithm used is sent with the digital signature, although this isn't shown in the figure 3.3.) Finally, the receiving software compares the new hash against the original hash. If the two hashes match, the data has not changed since it was signed. If they don't match, the data may have been tampered with since it was signed, or the signature may have been created with a private key that doesn't correspond to the public key presented by the signer.

If the two hashes match, the recipient can be certain that the public key used to decrypt the digital signature corresponds to the private key used to create the digital signature. Confirming the identity of the signer, however, also requires some way of confirming that the public key really belongs to a particular person or other entity. For a discussion of the way this works.

The significance of a digital signature is comparable to the significance of a handwritten signature. Once you have signed some data, it is difficult to deny doing so later--assuming that the private key has not been compromised or out of the owner's control. This quality of digital signatures provides a high degree of non-repudiation--that is, digital signatures make it difficult for the signer to deny having signed the data. In some situations, a digital signature may be as legally binding as a handwritten signature.

3.8 Attacks on Ciphers

Here the different kinds of possible attacks what have been observed so for and can be expected are explained in detail.

3.8.1 Exhaustive Key Search

Exhaustive key search, or brute-force search, is the basic technique of trying every possible key in turn until the correct key is identified. To identify the correct key it may be necessary to possess a plaintext and its corresponding ciphertext, or if the plaintext has some recognizable characteristic, ciphertext alone might suffice. Exhaustive key search can be mounted on any cipher and sometimes a weakness in the key schedule of the cipher can help improve the efficiency of an exhaustive key search attack. Advances in technology and computing performance will always make exhaustive key search an increasingly practical attack against keys of a fixed length. When DES was designed, it was generally considered secure against exhaustive key search without a vast financial investment in hardware. Over the years, this line of attack will become increasingly attractive to a potential adversary.

While the 56-bit key in DES now only offers a few hours of protection against exhaustive search by a modem dedicated machine, the current rate of increase in computing power is such that 80-bit key can be expected to offer the same level of protection against exhaustive key search in 18 years time as DES does today.

3.8.2 Differential Cryptanalysis

Differential cryptanalysis is a type of attack that can be mounted on iterative block ciphers. Differential cryptanalysis is basically a chosen plaintext attack and relies on an analysis of the evolution of the differences between two related plaintexts as they are encrypted under the same key. By careful analysis of the available data, probabilities can be assigned to each of the possible keys and eventually the most probable key is identified as the correct one.

Differential cryptanalysis has been used against a great many ciphers with varying degrees of success. In attacks against DES, its effectiveness is limited by what was very careful design of the S-boxes during the design of DES. Differential cryptanalysis has also been useful in attacking other cryptographic algorithms such as hash functions.

3.8.3 Linear Cryptanalysis

Linear cryptanalysis is a known plaintext attack and uses a linear approximation to describe the behavior of the block cipher. Given sufficient pairs of plaintext and corresponding ciphertext, bits of information about the key can be obtained and increased amounts of data will usually give a higher probability of success. There have been a variety of enhancements and improvements to the basic attack. Differential-linear cryptanalysis is an attack which combines elements of differential cryptanalysis with those of linear cryptanalysis. A linear cryptanalytic attack using multiple approximations might allow for a reduction in the amount of data required for a successful attack.

3.8.4 Weak Key for a Block Cipher

Weak keys are secret keys with a certain value for which the block cipher in question will exhibit certain regularities in encryption or, in other cases, a poor level of encryption. For instance, with DES there are four keys for which encryption is exactly the same as decryption. This means that if one were to encrypt twice with one of these weak keys, then the original plaintext would be recovered. For IDEA there is a class of keys for which cryptanalysis is greatly facilitated and the key can be recovered. However, in both these cases, the number of weak keys is such a small fraction of all possible keys that the chance of picking one at random is exceptionally slight. In such cases, they pose no significant threat to the security of the block cipher when used for encryption.

Of course for other block ciphers, there might well be a large set of weak keys (perhaps even with the weakness exhibiting itself in a different way) for which the chance of picking a weak key is too large for comfort. In such a case, the presence of weak keys would have an obvious impact on the security of the block cipher.

3.8.5 Algebraic Attacks

Algebraic attacks are a class of techniques which rely for their success on some block cipher exhibiting a high degree of mathematical structure. For instance, it is conceivable that a block cipher might exhibit what is termed a group structure. If this were the case, then encrypting a plaintext under one key and then encrypting the result under another key would always be equivalent to single encryption under some other single key. If so, then the block cipher would be considerably weaker, and the use of multiple encryptions would offer no additional security over single encryption. For most block ciphers, the question of whether they form a group is still open. For DES, however, it is known that the cipher is not a group. There are a variety of other concerns with regards to algebraic attacks.

3.8.6 Data Compression Used With Encryption

Data compression removes redundant character strings in a file. This means that the compressed file has a more uniform distribution of characters. In addition to providing shorter plaintext and ciphertext, which reduces the amount of time needed to encrypt, decrypt and transmit a file, the reduced redundancy in the plaintext can potentially hinder certain cryptanalytic attacks.

By contrast, compressing a file after encryption is inefficient. The ciphertext produced by a good encryption algorithm should have an almost statistically uniform distribution of characters. As a consequence, a compression algorithm should be unable to find redundant patterns in such text and there will be little, if any, data compression. In fact, if a data compression algorithm is able to significantly compress encrypted text, then this indicates a high level of redundancy in the ciphertext which, in turn, is evidence of poor encryption.

3.8.7 When an Attack Become Practical

There is no easy answer to this question since it depends on many distinct factors. Not only must the work and computational resources required by the cryptanalyst be reasonable, but the amount and type of data required for the attack to be successful must also be taken into account. One classification distinguishes among cryptanalytic attacks according to the data they require in the following way: chosen plaintext or chosen ciphertext, known plaintext, and ciphertext-only. This classification is not particular to secret-key ciphers and can be applied to cryptanalytic attacks on any cryptographic function. A chosen plaintext or chosen ciphertext attack gives the cryptanalyst the greatest freedom in analyzing a cipher. The cryptanalyst chooses the plaintext to be encrypted and analyzes the plaintext together with the resultant ciphertext to derive the secret key. Such attacks will, in many circumstances, be difficult to mount but they should not be discounted. A known plaintext attack is more useful to the cryptanalyst than a chosen plaintext attack (with the same amount of data) since the cryptanalyst now requires a certain numbers of plaintexts and their corresponding ciphertexts without specifying the values of the plaintexts. This type of information is presumably easier to collect. The most practical attack, but perhaps the most difficult to actually discover, is a ciphertext-only attack. In such an attack, the cryptanalyst merely intercepts a number of encrypted messages and subsequent analysis somehow reveals the key used for encryption. Note that some knowledge of the statistical distribution of the plaintext is required for a ciphertext-only attack to succeed.

An added level of sophistication to the chosen text attacks is to make them adaptive. By this we mean that the cryptanalyst has the additional power to choose the text that is to be encrypted or decrypted after seeing the results of previous requests. The computational effort and resources together with the amount and type of data required are all important features in assessing the practicality of some attack.

3.9 Strong Password-Only Authenticated Key Exchange

A new simple password exponential key exchange method (SPEKE) is described. It belongs to an exclusive class of methods which provide authentication and key establishment over an insecure channel using only a small password, without risk of off-line dictionary attack. SPEKE and the closely-related Diffie-Hellman Encrypted Key Exchange (DH-EKE) are examined in light of both known and new attacks, along with sufficient preventive constraints. Although SPEKE and DH-EKE are similar, the constraints are different. The class of strong password-only methods is compared to other authentication schemes. Benefits, limitations, and tradeoffs between efficiency and security are discussed. These methods are important for several uses, including replacement of obsolete systems, and building hybrid two-factor systems where independent password-only and key-based methods can survive a single event of either key theft or password compromise.

It seems paradoxical that small passwords are important for strong authentication. Clearly, cryptographically large passwords would be better, if only ordinary people could remember them. Password verification over an insecure network has been a particularly tough problem, in light of the ever-present threat of dictionary attack. Password problems have been around so long that many have assumed that strong remote authentication using only a small password is impossible. In fact, it can be done. In this paper we outline the problem, and describe a new simple password exponential key exchange, SPEKE, which performs strong authentication, over an insecure channel, using only a small password. That a small password can accomplish this alone goes against common wisdom. This is not your grandmother's network login. We compare SPEKE to the closely-related Diffie-Hellman Encrypted Key Exchange, and review the potential threats and countermeasures in some detail. We show that previously-known and new attacks against both methods are dissatisfied when proper constraints are applied. These methods are broadly useful for authentication in many applications: bootstrapping new system installations, cellular phones or other keypad systems, diskless workstations, user-to-user applications, multi-factor password + key systems, and for upgrading obsolete password systems. More generally, they are needed anywhere that prolonged key storage is risky or impractical, and where the communication channel may be insecure.

3.9.1 The Remote Password Problem

Ordinary people seem to have a fundamental inability to remember anything larger than a small secret. Yet most methods of remote secret-based authentication presume the secret to be large. We really want to use an easily memorized small secret password, and not are susceptible to dictionary attack. We make a clear distinction between passwords and keys: Passwords must be memorized, and are thus small, while keys can be recorded, and can be much larger. The problem is that most methods need keys that are too large to be easily remembered. User-selected passwords are often confined to a very small, easily searchable space, and attempts to increase the size of the space just make them hard to remember. Bank-card PIN codes use only 4-digits to remove even the temptation to write them down. A ten-digit phone number has about 30 bits, which compels many people to record them. Meanwhile, strong symmetric keys need 60 bits or more, and nobody talks about memorizing public-keys. It is also fair to assume that a memorizable password belongs to a brute-force searchable space. With ever-increasing computer power, there is a growing gap between the size of the smallest safe key and the size of the largest easily remembered password.

The problem is compounded by the need to memorize multiple passwords for different purposes. One example of a small-password-space attack is the verifiable plain-text dictionary attack against login. A general failure of many obsolete password methods is due to presuming passwords to be large. We assume that any password belongs to a cryptographically-small space, which is also brute-force searchable with a modest effort. Large passwords are arguably weaker since they can't be memorized.

So why do we bother with passwords? A pragmatic reason is that they are less expensive and more convenient than smart-cards and other alternatives. A stronger reason is that, in a well-designed and managed system, passwords are more resistant to theft than persistent stored keys or carry-around tokens. More generally, passwords represent something you know, one of the "big three" categories of factors in authentication.

3.9.2 Characteristics of Strong Password-only Methods

We now define exactly what we mean by strong password-only remote authentication. We first list the desired characteristics for these methods, focusing on the case of user-to-host authentication. Both SPEKE and DH-EKE have these distinguishing characteristics.

1. Prevent off-line dictionary attack on small passwords.
2. Survive on-line dictionary attack.
3. Provide mutual authentication.
4. Integrated key exchange.
5. User needs no persistent recorded.
6. (a) Secret data, or
   (b) Sensitive host-specific data.

Since we assume that all passwords are vulnerable to dictionary attack, given the opportunity, we need to remove the opportunities. On-line dictionary attacks can be easily detected, and thwarted, by counting access failures. But off-line dictionary attack presents a more complex threat. These attacks can be made by someone posing as a legitimate party to gather information, or by one who monitors the messages between two parties during a legitimate valid exchange. Even tiny amounts of information "leaked" during an exchange can be exploited. The method must be immune to such off-line attack, even for tiny passwords. This is where SPEKE and DH-EKE excel.

3.3 Authentication Confirms an Identity

Authentication is the process of confirming an identity. In the context of network interactions, authentication involves the confident identification of one party by another party. Authentication over networks can take many forms. Certificates are one way of supporting authentication.

Client and server authentication are not the only forms of authentication that certificates support. For example, the digital signature on an email message, combined with the certificate that identifies the sender, provide strong evidence that the person identified by that certificate did indeed send that message. Similarly, a digital signature on an HTML form, combined with a certificate that identifies the signer, can provide evidence, after the fact, that the person identified by that certificate did agree to the contents of the form. In addition to authentication, the digital signature in both cases ensures a degree of non-repudiation--that is, a digital signature makes it difficult for the signer to claim later not to have sent the email or the form.

Client authentication is an essential element of network security within most intranets or extranets. The sections that follow contrast two forms of client authentication:

- Password-Based Authentication. Almost all server software permits client authentication by means of a name and password. For example, a server might require a user to type a name and password before granting access to the server. The server maintains a list of names and passwords; if a particular name is on the list, and if the user types the correct password, the server grants access.
- Certificate-Based Authentication. Client authentication based on certificates is part of the SSL protocol. The client digitally signs a randomly generated piece of data

4. DES OVER SECURE CHANNEL

4.1 Overview

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." This chapter explains the various steps involved in DES-encryption, illustrating each step by means of a simple example. To understand DES easily, it better to understand first simplified DES (S_DES).

4.2 Simplified DES (S_DES)

S-DES is a simplified version of the well-known DES (Data Encryption Standard) algorithm .It closely resembles the real thing, with smaller parameters, to facilitate operation by hand for pedagogical purposes. It was designed by Edward Schaefer as a teaching tool to understand DES that has similar properties and structure but with much smaller parameters than DES. Figure 4.1 illustrate the simplified DES scheme. The programming of this algorithm will be in next chapter which will be an implementation of S_DES.

The S_DES encryption algorithm takes an 8-bit block of plaintext (example: 11001010) and a 10-bit key as input and produces an 8-bit block of ciphertext as output .the S_DES decryption algorithm takes an 8-bit block of ciphertext and the same 10-bit key used to produce that ciphertext as input and produces the original 8-bit block of plaintext.

The encryption algorithm involves five functions: an initial permutation (IP); a complex function labeled f_K, which involves both permutation and substitution operations and depends on a key input; a simple permutation function that switches (SW) the two halves of the data; the function f_K again, and finally a permutation function that is the inverse of initial permutation (IP^{-1}). The use of multiple stages of permutation and substitution results in a more complex algorithm, which increases the difficulty of cryptanalysis. The function f_K takes as input not only the data passing through the encryption algorithm, but also an 8-bit key. The algorithm could have been designed to work with a 16-bit key, consisting of two 8-bit subkeys, one used for each occurrence of f_K. Alternatively, a single 8-bit key could have been used, with the same key used twice in the algorithm. A compromise is to use a 10-bit key from which two 8-bit subkeys are generated, addicted in figure 4.1. In this case, the key is first subjected to permutation (P10). Then a shift operation is performed. The output of the shift operation then passes through a permutation function that produces an 8-bit output (P8) for the first subkey (K1). The output of shift operation also feeds into another shift and another instance of P8 to produce the second subkey (K2).

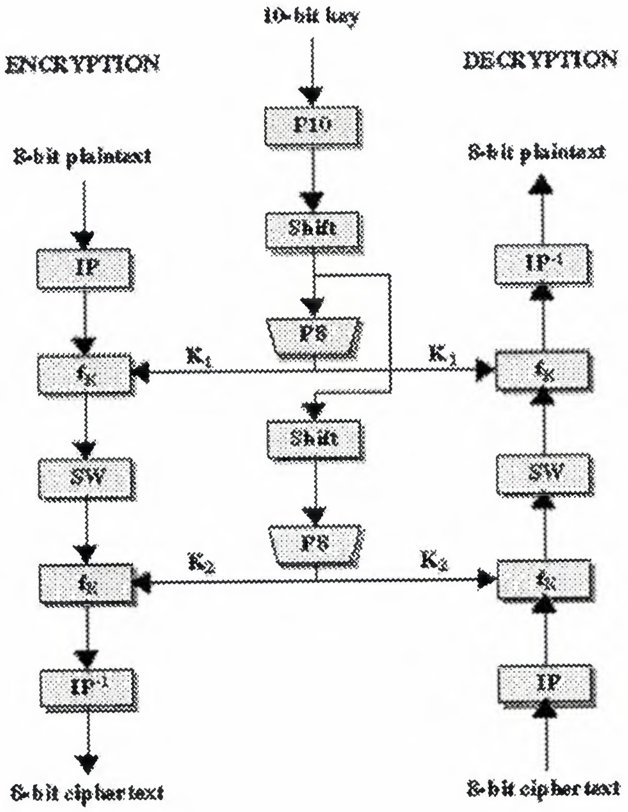


Figure 4.1 S_DES scheme

In f_K the rightmost 4 bits are passed through unchanged, and the leftmost 4 bits are "mangled" by the non-invertible function F:

$f_K(L,R)$ = L XOR F(R,K_i), R -- encrypt or decrypt

E/P = { 4, 1, 2, 3, 2, 3, 4, 1}

P4 = { 2, 4, 3, 1}

S0 = 1 0 3 2   S1 = 0 1 2 3
       3 2 1 0   2 0 1 3
       0 2 1 3   3 0 1 0
       3 1 3 2   2 1 0 3

$n_1n_2n_3n_4$ then $S_i[n_1n_4][n_2n_3]$

Example:

R = 1010

E/P 0101 0101

K1 = 1010 0100

XOR 1111 0001

S0[11][11] = 10   S1[01][00] = 10   -> P4 = 0011

4.2.1 Subkey generation

As in DES, the initial and final permutations, which are fixed and independent of the key, provide no real security benefit, but make the algorithm slow if implemented in software.

First, produce two subkeys K_1 and K_2:

K_1=P8(LS(P10(key)))

K_2 = P8(LS(LS(P10(key))))

where P10($k_1k_2k_3k_4k_5k_6k_7k_8k_9k_{10}$) = $k_3k_5k_2k_7k_4k_{10}k_1k_9k_8k_6$.

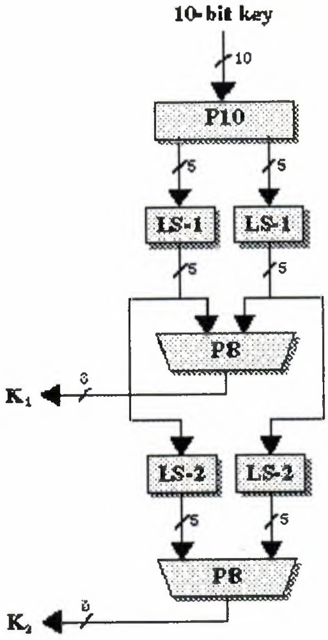


Figure 4.2 key Generation of S_DES

The 10-bit key is transformed into two 8-bit sub-keys K1 and K2.

Example:

P10 = { 3, 5, 2, 7, 4, 10, 1, 9, 8, 6}

P8 = { 6, 3, 7, 4, 8, 5, 10, 9}

K = 10100 00010

P10 = 10000 01100

LS-1 00001 11000 -> P8 -> K1 = 1010 0100

LS-2 00100 00011 -> P8 -> K2 = 0100 0011

4.2.2 Relation with DES

SDES is a simplification of a real algorithm. DES operates on 64 bit blocks, and uses a key of 56 bits, from which sixteen 48-bit subkeys are generated. There is an initial permutation (IP) of 56 bits followed by a sequence of shifts and permutations of 48 bits. F acts on 32 bits.

ciphertext = $IP^{-1}(f_{K16}(SW(f_{K15}(. . . (SW(f_{K1}(IP(plaintext)))). . .))))$

4.3 History of DES

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 accumulations 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 ("With Data Encryption, Scents Are Safe at IFF," Computerworld 14, No. 21, 95 (1980).) Meanwhile, the banking industry, which is the largest user of encryption outside 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. K = 00010011 00110100 01010111 01111001 10011011 10111100 11011111 11110001.

4.4 How DES Works in Detail

DES is a block cipher meaning it operates on plaintext blocks of a given size (64-bits) and returns cipher text 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.)

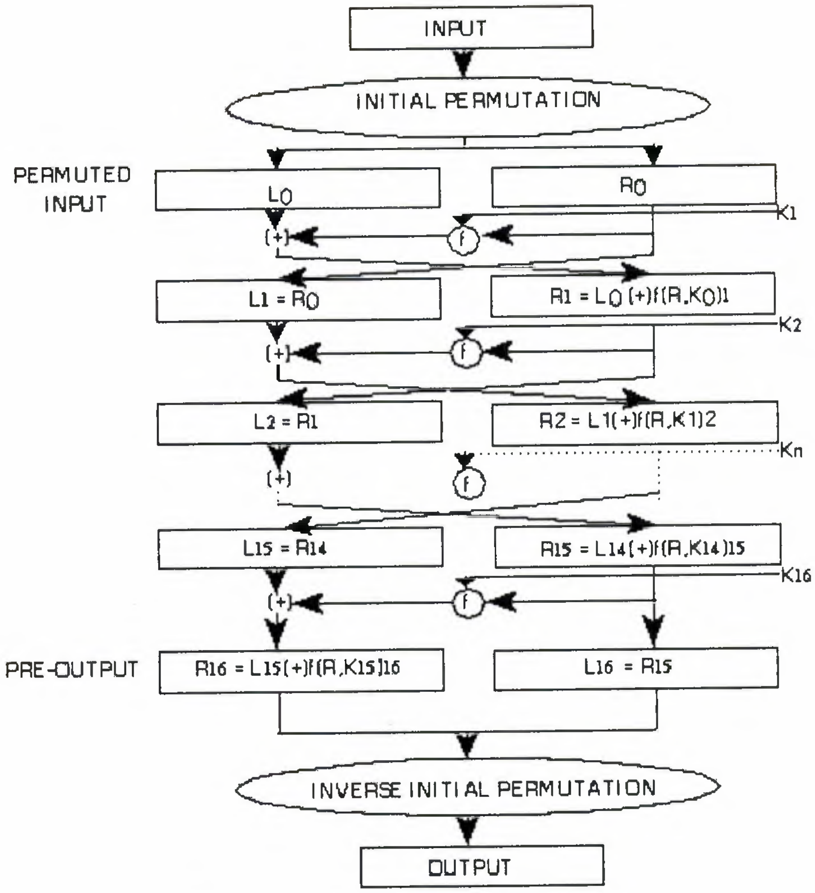


Figure 4.3 DES Encryption Structure

Example: 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 = 0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111

L = 0000 0001 0010 0011 0100 0101 0110 0111

R = 1000 1001 1010 1011 1100 1101 1110 1111

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 make subkeys.

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 = 00010011 00110100 01010111 01111001 10011011 10111100 11011111 11110001

The DES algorithm uses the following steps:

4.4.1 Step 1 find 16 subkeys, each of which is 48-bits long.

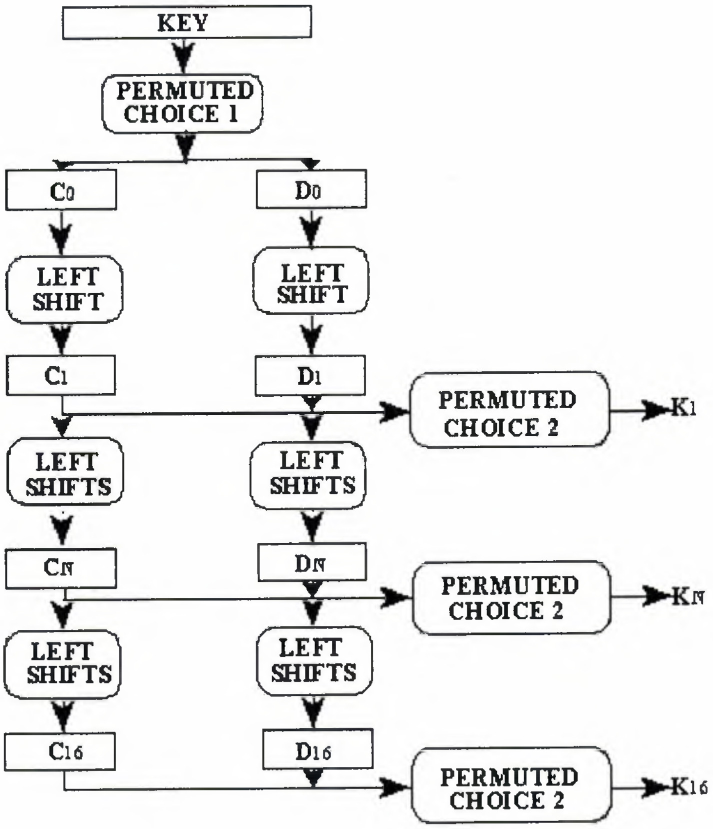


Figure 4.4 DES Key Setup

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.

Table 4.1 Permutation of 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 = 00010011 00110100 01010111 01111001 10011011 10111100 11011111 11110001

We get the 56-bit permutation

K+ = 1111000 0110011 0010101 0101111 0101010 1011001 1001111 0001111

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

C_0= 1111000 0110011 0010101 0101111

D_0 = 0101010 1011001 1001111 0001111

With C_0 and D_0 defined, we now create sixteen blocks C_n and D_n, 1<=n<=16. Each pair of blocks C_n and D_n is formed from the previous pair C_{n-1} and D_{n-1}, respectively, for 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 (The result shown in table 5.2).

Table 4.2

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, C_3 and D_3 are obtained from C_2 and D_2, respectively, by two left shifts, and C_{16} and D_{16} are obtained from C_{15} and D_{15}, respectively, by one left shift. In all cases, by a single left shift is meant 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 C_0 and D_0 we obtain:

C_0 = 1111000011001100101010101111

D_0 = 0101010101100110011110001111

C_1 = 1110000110011001010101011111

D_1 = 1010101011001100111100011110

C_2 = 1100001100110010101010111111

D_2 = 0101010110011001111000111101

C_3 = 0000110011001010101011111111

D_3 = 0101011001100111100011110101

C_4 = 0011001100101010101111111100

D_4 = 0101100110011110001111010101

C_5 = 1100110010101010111111110000

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.

Table 4.3 shows the result of second permutation

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 32th bit of C_nD_n.

Example: For the first key we have C_1D_1 = 1110000 1100110 0101010 1011111 1010101 0110011 0011110 0011110

This, after we apply the permutation PC-2, becomes

K_1 = 000110 110000 001011 101111 111111 000111 000001 110010

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.

4.4.2 Step 2: Encode 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.

Table4.4 First permutation of the message

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 = 0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111

IP = 1100 1100 0000 0000 1100 1100 1111 1111 1111 0000 1010 1010 1111 0000 1010 1010

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. 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 = 000110 110000 001011 101111 111111 000111 000001 110010

L_1 = R_0 = 1111 0000 1010 1010 1111 0000 1010 1010

$R_1 = L_0 + f(R_0,K_1)$

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 $E(R_{n-1})$ has a 32 bit input block, and a 48 bit output block.

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:

Table 4.5 E Bit Selection

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 $E(R_{n-1})$ are the bits in positions 32, 1 and 2 of R_{n-1} while the last 2 bits of $E(R_{n-1})$ are the bits in positions 32 and 1.

Example: We calculate $E(R_0)$ from R_0 as follows:

R_0 = 1111 0000 1010 1010 1111 0000 1010 1010

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

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

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

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

Example: For K_1 , $E(R_0)$, we have

K_1 = 000110 110000 001011 101111 111111 000111 000001 110010

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

K_1+$E(R_0)$ = 011000 010001 011110 111010 100001 100110 010100 100111.

We have not yet finished calculating the function f . To this point we have expanded R_{n-1} from 32 bits to 48 bits, using the selection table, and XORed the result with the key 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:

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

Where each B_i is a group of six bits. We now calculate

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

Where $S_i(B_i)$ referrers to the output of the *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 S_1 is shown and explained n table 5.6:

Table 4.6 Simulation Table

S1

Row	**Column Number**															
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

f 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" giving 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 $S_1,\ldots,S_8$ are the following in table 5.7:

Table 4.7

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: $K_1 + E(R_0)$ = 011000 010001 011110 111010 100001 100110 010100 100111.

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

$S_1(B_1)S_2(B_2)S_3(B_3)S_4(B_4)S_5(B_5)S_6(B_6)S_7(B_7)S_8(B_8)$ = 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:

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

The permutation P is defined in the table5.8. P yields a 32-bit output from a 32-bit input by permuting the bits of the input block.

Table 4.8 P results

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:

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

we get

f= 0010 0011 0100 1010 1010 1001 1011 1011

$R_1 = L_0 + f(R_0 , K_1)$

= 1100 1100 0000 0000 1100 1100 1111 1111

+ 0010 0011 0100 1010 1010 1001 1011 1011

= 1110 1111 0100 1010 0110 0101 0100 0100

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 table5.9:

Table 4.9 final permutation

IP^{-1}

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

That is, the output of the algorithm has bit 40 of the pre output 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

$R_{16}L_{16}$ = 00001010 01001100 11011001 10010101 01000011 01000010 00110010 00110100

IP^{-1} = 10000101 11101000 00010011 01010100 00001111 00001010 10110100 00000101

which in hexadecimal format is

85E813540F0AB405.

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

C = 85E813540F0AB405.

Decryption is simply the inverse of encryption, following the same steps as above, but reversing the order in which the subkeys are applied.

4.4.3 DES Modes of Operation

The DES algorithm turns a 64-bit message blocks M into a 64-bit cipher block C. If each 64-bit block is encrypted individually, then the mode of encryption is called Electronic Code Book (ECB) mode. There are two other modes of DES encryption, namely Chain Block Coding (CBC) and Cipher Feedback (CFB), which make each cipher block dependent on all the previous messages blocks through an initial XOR operation which explained previous

4.4.4 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" where 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 cipher text "0000000000000000". If the cipher text 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, "596F7572206C6970 732061726520736D 6F6F746865722074 68616E2076617365 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:
"596F7572206C6970 732061726520736D 6F6F746865722074 68616E2076617365 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: "C0999FDDE378D7ED 727DA00BCA5A84EE 47F269A4D6438190 9DD52F78F5358499 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". (Think how much better off Bill Clinton would be today, if Monica Lewinsky had used encryption on her Pentagon computer!)

4.5 Cracking DES

Before DES was adopted as a national standard, during the period NBS was soliciting comments on the proposed algorithm, the creators of public key cryptography, Martin Hellman and Whitfield Diffie, registered some objections to the use of DES as an encryption algorithm. Hellman wrote: "Whit Diffie and I have become concerned that the proposed data encryption standard, while probably secure against commercial assault, may be extremely vulnerable to attack by an intelligence organization" (letter to NBS, October 22, 1975).

Diffie and Hellman then outlined a "brute force" attack on DES. (By "brute force" is meant that you try as many of the 2^56 possible keys as you have to before decrypting the cipher text into a sensible plaintext message.) They proposed a special purpose "parallel computer using one million chips to try one million keys each" per second, and estimated the cost of such a machine at $20 million.

A brute force attack remains the most promising approach. For SDES, it is trivial; there are only 1024 keys. For DES it is much harder; there are 7×10^{16} keys. This would take over 2000 years if you checked each key in one microsecond. Michael Wiener designed a DES cracker with millions of specialized chips on specialized boards and racks [1, p.153]. He concluded in 1995 that for $1 million, a machine could be built that would crack a 56-bit DES key in 7 hours. Schneier estimates that this would be doable for $100,000 in 2000, so we might extrapolate to $10,000 in 2005. For these reasons, algorithms based on 56-bit keys, like DES, are no longer thought secure. Schneier: "insist on at least 112 bit keys."

Fast forward to 1998. Under the direction of John Gilmore of the EFF, a team spent $220,000 and built a machine that can go through the entire 56-bit DES key space in an average of 4.5 days. On July 17, 1998, they announced they had cracked a 56-bit key in 56 hours. The computer, called Deep Crack, uses 27 boards each containing 64 chips, and is capable of testing 90 billion keys a second.

Despite this, as recently as June 8, 1998, Robert Litt, principal associate deputy attorney general at the Department of Justice, denied it was possible for the FBI to crack DES: "Let me put the technical problem in context: It took 14,000 Pentium computers working for four months to decrypt a single message . . . . We are not just talking FBI and NSA [needing massive computing power], we are talking about every police department." Responded cryptography expert Bruce Schneider: " . . . the FBI is either incompetent or lying, or both." Schneider went on to say: "The only solution here is to pick an algorithm with a longer key; there isn't enough silicon in the galaxy or enough time before the sun burns out to brute- force triple-DES" (Crypto-Gram, Counterpane Systems, August 15, 1998).

4.6 Triple-DES

Triple-DES is just DES with two 56-bit keys applied. Given a plaintext message, the first key is used to DES- encrypt the message. The second key is used to DES-decrypt the encrypted message. (Since the second key is not the right key, this decryption just scrambles the data further.) The twice-scrambled message is then encrypted again with the first key to yield the final cipher text. This three-step procedure is called triple-DES.

Triple-DES is just DES done three times with two keys used in a particular order. (Triple-DES can also be done with three separate keys instead of only two. In either case the resultant key space is about 2^112.)

5. IMPLEMENTATION OF S_DES BY USING C LANGUAGE

5.1 Overview

This chapter explains software for Simplified Data Encryption Stranded (SDES) using C language. Simplified of DES is an educational rather than a secure encryption algorithm. It has similar properties and structure to DES with much smaller parameters. It was developed by Professor Edward Schaefer

5.2 Flow Chart of Software

The below diagram explain how the program work

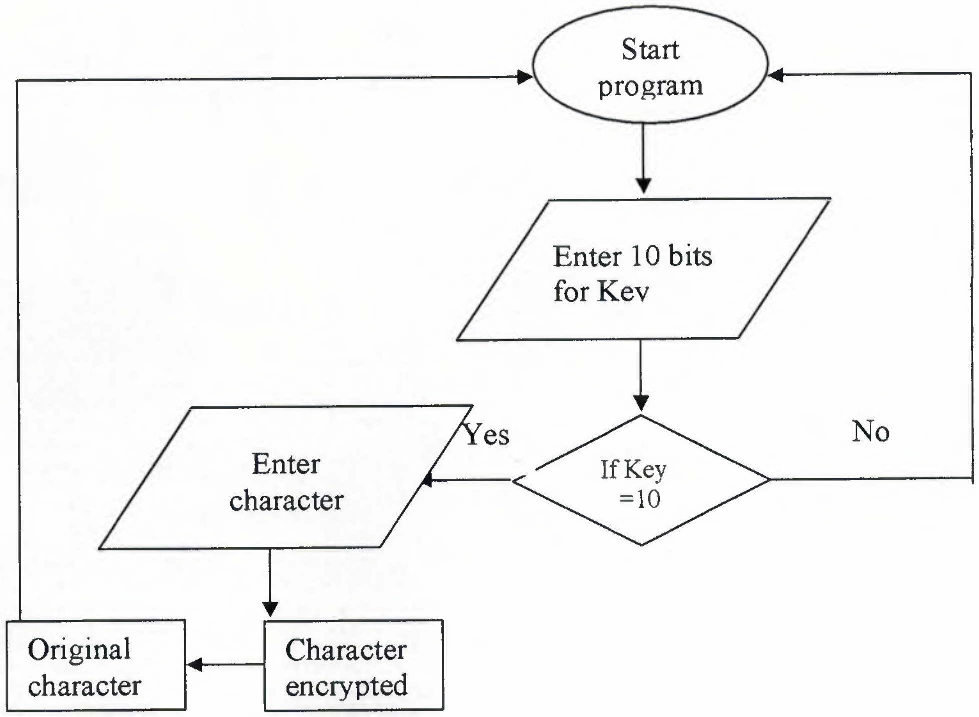
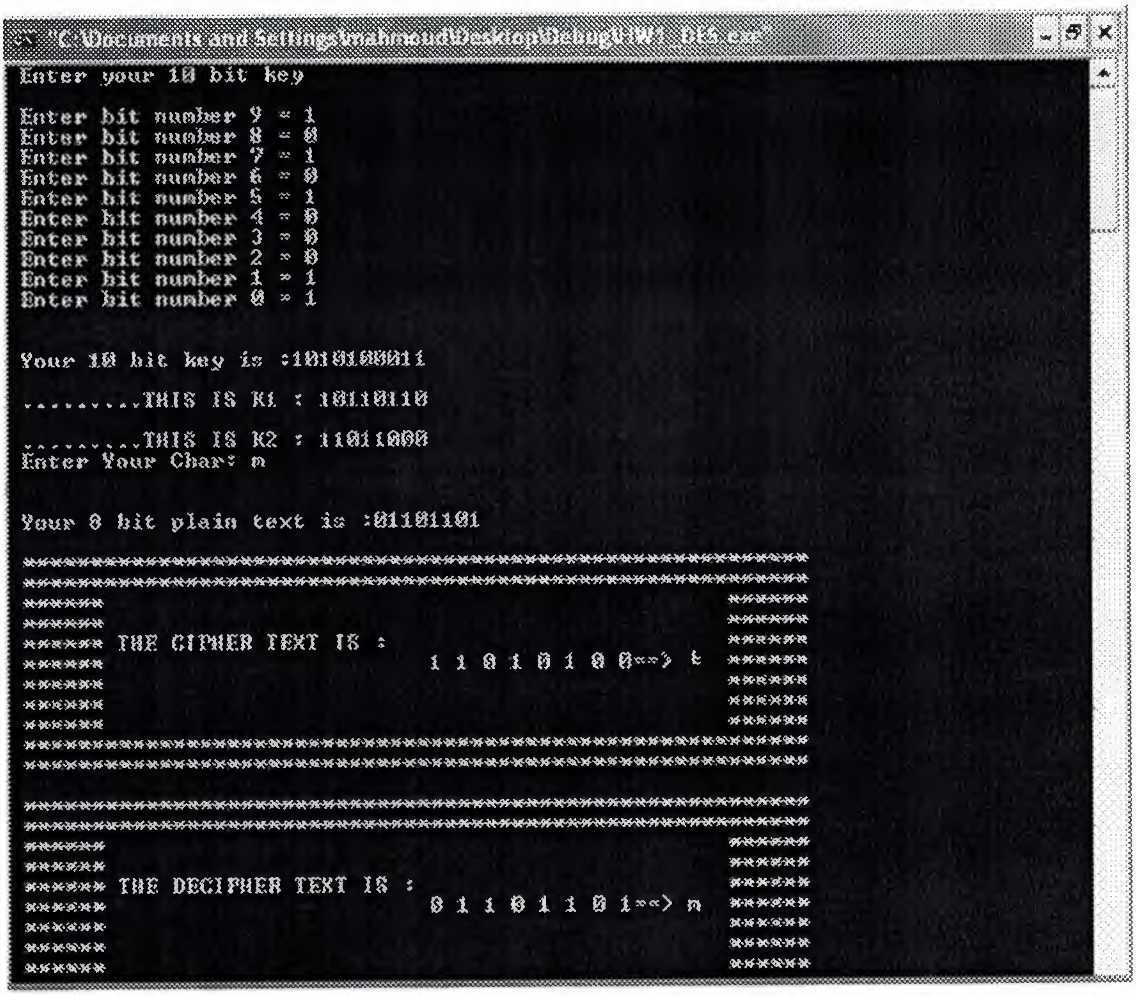


Figure 5.1 flowchart of S_DES

5.3 Encryption and Decryption Algorithms

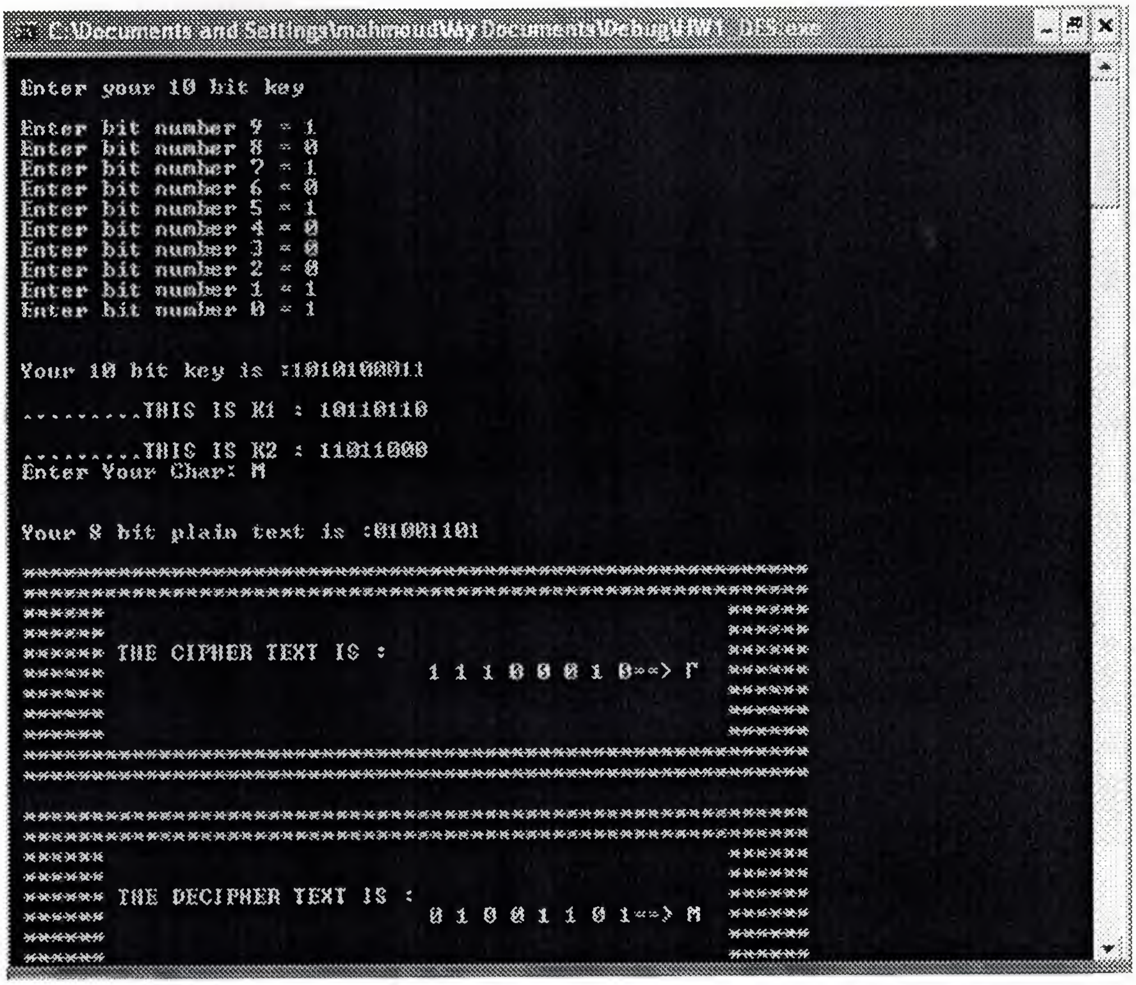
After the user enter the private key, the user will enter the character to encrypt, which describe in figure below, after encryption the character will decrypt to same character. The S_DES encryption algorithm takes an 8_bit block of plaintext (example 10111101) and a 10 _bit key as input and produces an 8_bit block of ciphertext as output. The S_DES decryption algorithm takes an 8_bit block of ciphertext and the same 10_bit key used to produce that ciphertext as input and produces the original 8 _bit block of plaintext. Note that the enter character of message is a letter, which the program changes it to ACSII which is 8_bit of binary.


"C:\Documents and Settings\mahmoud\Desktop\Debug\1W1_DES.exe"

```
Enter your 10 bit key

Enter bit number 9 = 1
Enter bit number 8 = 0
Enter bit number 7 = 1
Enter bit number 6 = 0
Enter bit number 5 = 1
Enter bit number 4 = 0
Enter bit number 3 = 0
Enter bit number 2 = 0
Enter bit number 1 = 1
Enter bit number 0 = 1

Your 10 bit key is :1010100011

..........THIS IS K1 : 10110110

..........THIS IS K2 : 11011000
Enter Your Char: m

Your 8 bit plain text is :01101101

**********************************************************
**********************************************************
******                                              ******
******                                              ******
****** THE CIPHER TEXT IS :                         ******
******                        1 1 0 1 0 1 0 0==> t  ******
******                                              ******
******                                              ******
******                                              ******
**********************************************************
**********************************************************

**********************************************************
**********************************************************
******                                              ******
******                                              ******
****** THE DECIPHER TEXT IS :                       ******
******                        0 1 1 0 1 1 0 1==> m  ******
******                                              ******
******                                              ******
******                                              ******
```

Figure 5.2

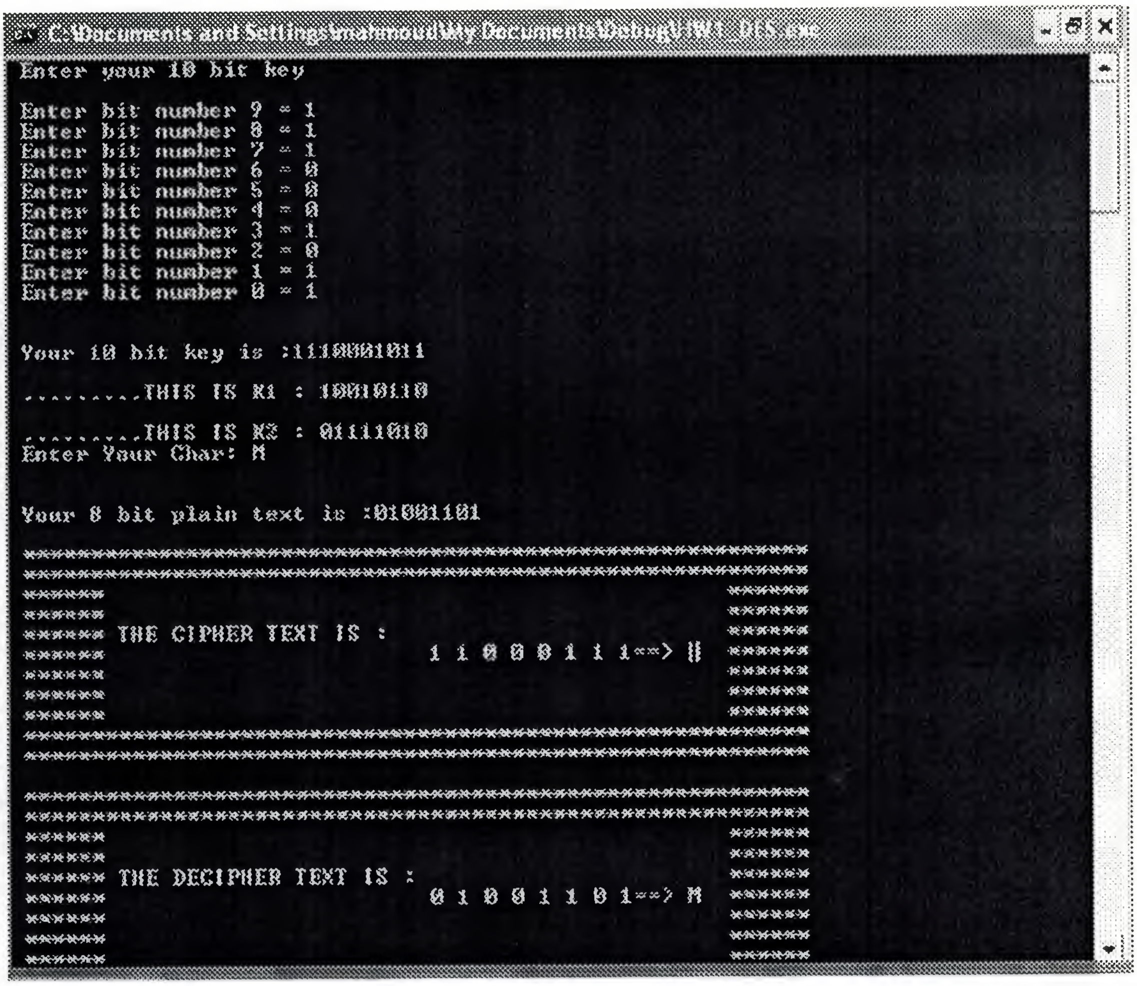
4.5 Examples of Encryption and Decryption



```
Enter your 10 bit key

Enter bit number 9 = 1
Enter bit number 8 = 0
Enter bit number 7 = 1
Enter bit number 6 = 0
Enter bit number 5 = 1
Enter bit number 4 = 0
Enter bit number 3 = 0
Enter bit number 2 = 0
Enter bit number 1 = 1
Enter bit number 0 = 1


Your 10 bit key is :1010100011

..........THIS IS K1 : 10110110

..........THIS IS K2 : 11011000
Enter Your Char: M

Your 8 bit plain text is :01001101

***********************************************************
***********************************************************
******                                               ******
******                                               ******
****** THE CIPHER TEXT IS :                          ******
******                       1 1 1 0 0 0 1 0--> Γ    ******
******                                               ******
******                                               ******
******                                               ******
***********************************************************
***********************************************************

***********************************************************
***********************************************************
******                                               ******
******                                               ******
****** THE DECIPHER TEXT IS :                        ******
******                       0 1 0 0 1 1 0 1--> M    ******
******                                               ******
******                                               ******
```

Figure 5.3

In this example the message is (M) but in the example of figure 5.2 the message is (m) and the same key is used for the both, but the ciphertext changes from (t) to (Γ).

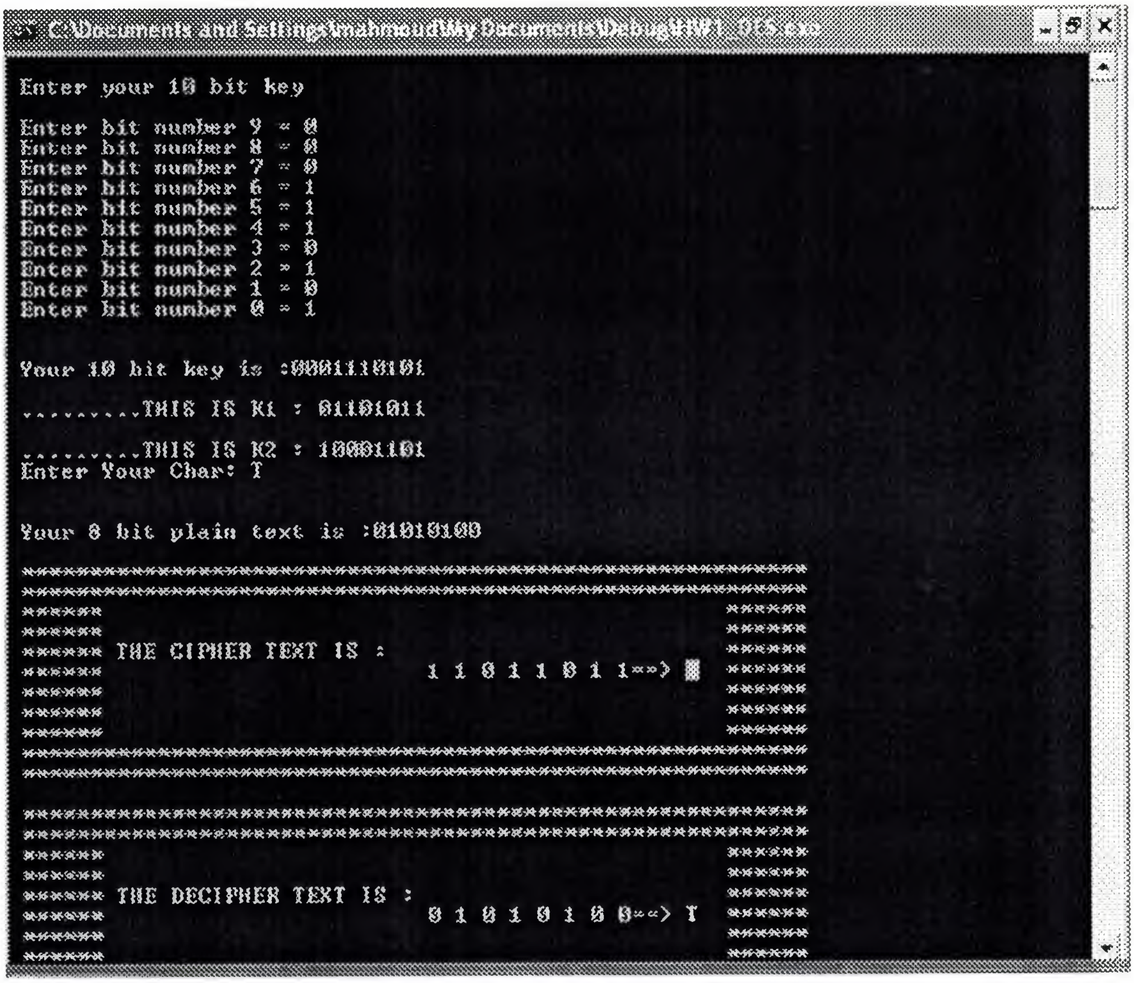


C:\Documents and Settings\mahmoud\My Documents\Debug\IW1_DES.exe

```
Enter your 10 bit key

Enter bit number 9 = 1
Enter bit number 8 = 1
Enter bit number 7 = 1
Enter bit number 6 = 0
Enter bit number 5 = 0
Enter bit number 4 = 0
Enter bit number 3 = 1
Enter bit number 2 = 0
Enter bit number 1 = 1
Enter bit number 0 = 1


Your 10 bit key is :1110001011

..........THIS IS K1 : 10010110

..........THIS IS K2 : 01111010
Enter Your Char: M


Your 8 bit plain text is :01001101

*****************************************************
*****************************************************
******                                         ******
******                                         ******
****** THE CIPHER TEXT IS :                    ******
******                      1 1 0 0 0 1 1 1==> ╟ ******
******                                         ******
******                                         ******
******                                         ******
*****************************************************
*****************************************************


*****************************************************
*****************************************************
******                                         ******
******                                         ******
****** THE DECIPHER TEXT IS :                  ******
******                      0 1 0 0 1 1 0 1==> M ******
******                                         ******
******                                         ******
******                                         ******
```

Figure 5.4

In this example as shown in the figure 5.4 the 10_bit key is changed to (1110001011), but the message is the same (M) but the cipher text is changed to (╟).


C:\Documents and Settings\mahmoud\My Documents\Debug\HW1_DES.exe

```
Enter your 10 bit key

Enter bit number 9 = 0
Enter bit number 8 = 0
Enter bit number 7 = 0
Enter bit number 6 = 1
Enter bit number 5 = 1
Enter bit number 4 = 1
Enter bit number 3 = 0
Enter bit number 2 = 1
Enter bit number 1 = 0
Enter bit number 0 = 1

Your 10 bit key is :0001110101

..........THIS IS K1 : 01101011

..........THIS IS K2 : 10001101
Enter Your Char: T

Your 8 bit plain text is :01010100

***********************************************************
***********************************************************
******                                               ******
******                                               ******
****** THE CIPHER TEXT IS :                          ******
******                        1 1 0 1 1 0 1 1--> █   ******
******                                               ******
******                                               ******
******                                               ******
***********************************************************
***********************************************************

***********************************************************
***********************************************************
******                                               ******
******                                               ******
****** THE DECIPHER TEXT IS :                        ******
******                        0 1 0 1 0 1 0 0--> T   ******
******                                               ******
******                                               ******
```

Figure 5.5

The key entered in figure 5.5 is (0001110101), the plaintext is T and the cipher text is (█).

4.6 Summary

The simplified Data Encryption Standard specifies a cryptographic algorithm that converts plaintext to ciphertext using a key .The same algorithm is used with the same key to convert ciphertext back to plaintext mix the data and key together. The goal is to completely scramble the data and key so that every bit of the ciphertext depends on every bit of the data and every bit of the key .After sufficient "rounds", there should be no correlation between the ciphertext and either the original data or key.

REFERENCES:

[1] Symmetric cryptographic system for data encryption by C. ADAMS in Apr 1996.

[2] Proceedings of the Internet Society Symposium on Network and Distributed System Security by IDUP and SPKM in1996.

[3] Random sources for cryptographic systems by G.B. AGNEW in 1988.

[4] An implementation for a fast public-key cryptosystem by G.B. AGNEW, R.C. MULLIN, I.M. ONYSZCHUK & S.A. VANSTONE in1991.

[5] A Secure Key Distribution System by N. ALEXANDRIS, M. BURMESTER & V. CHRISSIKOPOULOS.

[6] A Weakness in the 4.2BSD UNIX TCP/IP Software by R.T. Morris in 1985.

[7] Security Problems in the TCP/IP Protocol Suite Vol. 19 by S.M. Bellovin in April 1989.

[8] Handbook of Applied Cryptography by A. Menezes, P. van Oorschot, and S. Vanstone in 1996.

[9] Edward G. Amoroso," Fundamentals of Computer Security and Network Technology", Second Edition, Prentice Hall, May1994.

[10] MEDINA, M., "E-Mail Security - Public Key Distribution and Certification Paths," MSc Dissertation, Catholic University of Rio de Janeiro, 1996.

[11] Internet Mail Consortium, postings to PKIX mailing list, under the title "certificate path services," 15-16 October 1998, http://www.imc.org/ietf-pkix. Retrieved October 30,2003.

[12] ADAMS, A., Zuccherato, R., "Internet X.509 Public Key Infrastructure - Data Certification Server Protocols," IETF Draft, PKIX Working Group, <draft-ietf-pkix-dcs-00.txt>, September 1998.

[13] DANIELSSON, J., Westerlund, A., "KTH-KRB (Kerberos 4 from KTH)", Edition 1.0 for version 0.9.8, Kungliga Tekniska Högskolan - Royal Institute of Technology, Stockholm, Sweden, December 1997, http://www.pdc.kth.se/kth-krb. Retrived December 5,2003.

[14] University of Michigan, "Lightweight Directory Access Protocol," http://www.umich.edu/~dirsvcs/ldap. Retrieved November 15, 2003.

[15] Queensland University of Technology, "Oscar - DSTC's Public key Infrastructure Project," December 1998, http://oscar.dstc.qut.edu.au/. Retrieved December 5, 2003.

[16] Dorothy Denning April 3, 1998
http://www.cs.technion.ac.il/%7Ebiham/publications.html. Retrieved December 14, 2003

[17] J. Orlin Grabbe The DES Algorithm Illustrated
Homepage: http://orlingrabbe.com. Retrieved October 25, 2003.

[18] Data Encryption Standard, Federal Information Processing Standard (FIPS) Publication 46, National Bureau of Standards, U.S. Department of Commerce, Washington D.C. (January 1977).

[19] Miles E. Smid and Dennis K. Branstad, "The Data Encryption Standard: Past and Future," in Gustavus J. Simmons, ed., Contemporary Cryptography: The Science of Information Integrity, IEEE Press, 1992.

APPENDIX

```
// THIS PROGRAM PRESENTS S_DES ALGORITHM
// PREPARED BY: MAHMOUD ALLABADI
// REG. NUMBER: 20011073
// TO        : Prof.Dr.FAKHREDDIN MAMEDOV

#include<stdio.h>
#include<conio.h>

//Global variable's
int k1[8];
int k2[8];
int flag;

//Prototype
void find_keys();
void fk(int input[8]);
void permut(int array[],int len);
void shift_left(int array[],int len);
void E_P(int array[4]);
int S0(int array[4]);
int S1(int array[4]);
char convert_char(int array[8]);

void main(){
        char message;
        int plaintext[8];
        int array[8];
        int i,k;
        //clrscr();
        find_keys();

        printf("\n Enter Your Char: ");
        message=getchar();
        message=getchar();
        k=message;
        for(i=0;i<8;i++){plaintext[i]=(k%2);
                    k=k/2;
                         }


        printf("\n\n Your 8 bit plain text is :");
        for(i=7;i>=0;i--){array[i]=plaintext[i];
                          printf("%d",plaintext[i]);
                    }

        // IP of the 8 bit

        permut(array,8);
        flag=1;
        fk(array);

        permut(array,8);
        flag=2;
        //fk #2 after SW
        fk(array);
        //After IP-1
        permut(array,8);

        printf("\n\n
**********************************************************");
        printf("\n
**********************************************************");
        printf("\n ******                              ******");
        printf("\n ******                              ******");
        printf("\n ****** THE CIPHER TEXT IS :                 ******");
        printf("\n ******");
        printf("                    ");

        for(i=7;i>=0;i--)printf(" %d",array[i]);
        printf("==> %c ",convert_char(array));

        printf(" ******");
        printf("\n ******                              ******");
        printf("\n ******                              ******");
        printf("\n ******                              ******");
        printf("\n
**********************************************************");
        printf("\n
**********************************************************");

        //deciphering
        permut(array,8);
        fk(array);
        //switching
        permut(array,8);
        flag=1;
        fk(array);
        permut(array,8);
        printf("\n\n
**************************************************************");
        printf("\n
**************************************************************");
        printf("\n ******                              ******");
        printf("\n ******                              ******");
        printf("\n ****** THE DECIPHER TEXT IS :                 ******");
        printf("\n ******");
        printf("                ");

        for(i=7;i>=0;i--)printf(" %d",array[i]);
        printf("==> %c ",convert_char(array));
        printf(" ******");
        printf("\n ******                              ******");
        printf("\n ******                              ******");
        printf("\n ******                              ******");
        printf("\n
**************************************************************");
        printf("\n
**************************************************************\n");


        getche();
        }


//THE BIGIN OF THE FUNCTION find_keys()
//The function is to find k1 and k2
void find_keys(){
        int Ten_key[10],i;
        int Five_L[5];
        int Five_R[5];
        int Eight_bit[8];

        // reading the 10_bit key
        printf("\n Enter your 10 bit key\n\n");
        for(i=9;i>=0;i--){printf(" Enter bit number %d = ",i);
                        scanf("%d",&Ten_key[i]);
                        }
        printf("\n\n Your 10 bit key is :");
        for(i=9;i>=0;i--)printf("%d",Ten_key[i]);

        // permutation of the 10 bit
        permut(Ten_key,10);

        // devide the 10 bit to Right  And Left
        for(i=0;i<5;i++)Five_L[i]=Ten_key[i];
        for(i=4;i>=0;i--)Five_R[i]=Ten_key[i+5];

        // Shifting 1 step to Left

        shift_left(Five_R,5);
        shift_left(Five_L,5);

        //Combine it in 8 bit array
        for(i=0;i<4;i++)Eight_bit[i]=Five_L[i];
        for(i=0;i<4;i++)Eight_bit[i+4]=Five_R[i];

        //permutation to the Eight bit
        permut(Eight_bit,8);
        printf("\n\n .........THIS IS K1 : ");
        for(i=0;i<8;i++)k1[i]=Eight_bit[i];
        for(i=7;i>=0;i--)printf("%d",k1[i]);

        // Shifting 2 step to Left

        shift_left(Five_R,5);
        shift_left(Five_R,5);

        shift_left(Five_L,5);
        shift_left(Five_L,5);

        //Combine it in 8 bit array
        for(i=0;i<4;i++)Eight_bit[i]=Five_L[i];
        for(i=0;i<4;i++)Eight_bit[i+4]=Five_R[i];

        //permutation to the Eight bit

        permut(Eight_bit,8);
        printf("\n\n .........THIS IS K2 : ");
        for(i=0;i<8;i++)k2[i]=Eight_bit[i];
        for(i=7;i>=0;i--)printf("%d",k2[i]);


        }//THE END OF THE FUNCTION find_keys()


//THE BIGIN OF THE FUNCTION cipher()
//The function is to Encryption the massege

void fk(int input[8]){
        int Four_L[4];
        int Four_R[4];
        int i;


        // devide the 8 bit to Right  And Left
        for(i=0;i<4;i++){Four_L[i]=input[i];}
        for(i=3;i>=0;i--)Four_R[i]=input[i+4];

        E_P(Four_L);

        // now xor with Four_R
        for(i=0;i<4;i++){
                        if(Four_L[i]==Four_R[i])Four_L[i]=0;
                        else Four_L[i]=1;
                        }

        for(i=4;i<8;i++)input[i]=Four_L[i-4];

        }//THE END OF THE FUNCTION fk()



//the function for permutation
void permut(int array[],int len){
        int i,x;
        for(i=0;i<len/2;i++){
                          x=array[i];
                          array[i]=array[i+len/2];
                          array[i+len/2]=x;
                          }

                              }//END of fun permute()




//the function for shift left
void shift_left(int array[],int len){
        int i,x;

        x=array[len-1];
        for(i=len-1;i>0;i--)array[i]=array[i-1];
        array[0]=x;
```

```
                }//END of fun shift_left()


void E_P(int array[4]){
        int Eight[8];
        int Four_L2[4];
        int Four_R2[4];
        int i,val0,val1;

        Eight[7]=array[0];
        Eight[6]=array[3];
        Eight[5]=array[2];
        Eight[4]=array[1];
        Eight[3]=array[2];
        Eight[2]=array[1];
        Eight[1]=array[0];
        Eight[0]=array[3];


        if(flag==1){
                // now xor with k1
                for(i=0;i<8;i++){
                            if(k1[i]==Eight[i])Eight[i]=0;
                            else Eight[i]=1;
                               }
                }
        if(flag==2){
                // now xor with k2
                for(i=0;i<8;i++){
                            if(k2[i]==Eight[i])Eight[i]=0;
                            else Eight[i]=1;
                               }
                }

        // devide the 8 bit to Right  And Left
        for(i=0;i<4;i++)Four_L2[i]=Eight[i];
        for(i=3;i>=0;i--)Four_R2[i]=Eight[i+4];

        // Entering Four_R2 to S0
        val0=S0(Four_R2);

        // Entering Four_L2 to S1
        val1=S1(Four_L2);

        if((val0==0) && (val1==0)){
```

```
                              array[0]=0;
                              array[1]=0;
                              array[2]=0;
                              array[3]=0;
                              }
if((val0==0) && (val1==1)){
                              array[0]=1;
                              array[1]=0;
                              array[2]=0;
                              array[3]=0;
                              }
if((val0==0) && (val1==2)){
                              array[0]=0;
                              array[1]=1;
                              array[2]=0;
                              array[3]=0;
                              }
if((val0==0) && (val1==3)){
                              array[0]=1;
                              array[1]=1;
                              array[2]=0;
                              array[3]=0;
                              }
if((val0==1) && (val1==0)){
                              array[0]=0;
                              array[1]=0;
                              array[2]=1;
                              array[3]=0;
                              }
if((val0==1) && (val1==1)){
                              array[0]=1;
                              array[1]=0;
                              array[2]=1;
                              array[3]=0;
                              }
if((val0==1) && (val1==2)){
                              array[0]=0;
                              array[1]=1;
                              array[2]=1;
                              array[3]=0;
                              }
if((val0==1) && (val1==3)){
                              array[0]=1;
                              array[1]=1;
                              array[2]=1;
                              array[3]=0;
```

```
                      }
if((val0==2) && (val1==0)){
                      array[0]=0;
                      array[1]=0;
                      array[2]=0;
                      array[3]=1;
                      }
if((val0==2) && (val1==1)){
                      array[0]=1;
                      array[1]=0;
                      array[2]=0;
                      array[3]=1;
                      }
if((val0==2) && (val1==2)){
                      array[0]=0;
                      array[1]=1;
                      array[2]=0;
                      array[3]=1;
                      }
if((val0==2) && (val1==3)){
                      array[0]=1;
                      array[1]=1;
                      array[2]=0;
                      array[3]=1;
                      }
if((val0==3) && (val1==0)){
                      array[0]=0;
                      array[1]=0;
                      array[2]=1;
                      array[3]=1;
                      }
if((val0==3) && (val1==1)){
                      array[0]=1;
                      array[1]=0;
                      array[2]=1;
                      array[3]=1;
                      }
if((val0==3) && (val1==2)){
                      array[0]=0;
                      array[1]=1;
                      array[2]=1;
                      array[3]=1;
                      }
if((val0==3) && (val1==3)){
                      array[0]=1;
                      array[1]=1;
```

```
                                        array[2]=1;
                                        array[3]=1;
                                        }

        permut(array,4);

                }//END of fun E_P



int S0(int array[4]){
        int row,col,val0;
        int s0[4][4]={1,0,3,2,3,2,1,0,0,2,1,3,3,1,3,2};

        if((array[3]==0) && (array[0]==0))row=0 ;
        if((array[2]==0) && (array[1]==0))col=0 ;
        if((array[3]==0) && (array[0]==1))row=1 ;
        if((array[2]==0) && (array[1]==1))col=1 ;
        if((array[3]==1) && (array[0]==0))row=2 ;
        if((array[2]==1) && (array[1]==0))col=2 ;
        if((array[3]==1) && (array[0]==1))row=3 ;
        if((array[2]==1) && (array[1]==1))col=3 ;


        val0=s0[row][col];
        return val0;

           }//END of fun S0



int S1(int array[4]){
        int row,col,val1;
        int s1[4][4]={0,1,2,3,2,0,1,3,3,0,1,0,2,1,0,3};

        if((array[3]==0) && (array[0]==0))row=0 ;
        if((array[2]==0) && (array[1]==0))col=0 ;
        if((array[3]==0) && (array[0]==1))row=1 ;
        if((array[2]==0) && (array[1]==1))col=1 ;
        if((array[3]==1) && (array[0]==0))row=2 ;
        if((array[2]==1) && (array[1]==0))col=2 ;
        if((array[3]==1) && (array[0]==1))row=3 ;
        if((array[2]==1) && (array[1]==1))col=3 ;
        val1=s1[row][col];
        return val1;
                }//END of fun S1

char convert_char(int array[8]){
                                char ch;
                                int i,f=0,d,z;

                                for(i=0;i<8;i++){z=1;
                                        if(array[i]==1){
                                                for(d=1;d<=i;d++)z*=2;
                                                                        }
                                        else z=0;
                                        f+=z;
                                                                }
                                ch=f;
                                return ch;
}
```

CONCLUSION

The most efficient way to provide security for communication between parties within a communication network is the use of cryptography. However, in order for this mechanism to keep information secure, safe ways of distributing keys must exist.

Data encryption devices are deployed within a vast number of different critical data communication applications. DES is the most common of encryption methods which devices with their extensive features and cost effectiveness compete effectively; it enhances performance and security within communication applications.

Since its adoption as federal standard, there have been lingering concerns about the level of security provided by DES. These concerns, by and large, fall into two areas: key size and the nature of the algorithm.

Today's connected society requires secure data encryption devices to preserve data privacy and authentication in critical applications. One of the several data encryption types, Data Encryption Standard (DES) has emerged to be the most commonly used in varying applications. Data encryption is used pervasively in today's connected society.