

NEAR EAST UNIVERSITY



Faculty of Engineering

Department of Computer Engineering

PUBLIC KEY WITH SECURITY NETWORK

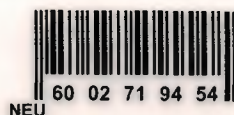
Graduation Project

COM- 400

Student: Zaki Abdallah (20002179)

Supervisor: Asst. Prof. Dr. Firudin Muradov

Nicosia – 2005





ACKNOWLEDGEMENT

First of all, I want to pay my regards and to express my sincere gratitude to my supervisor Ass. Prof. Dr Firudin Muradov. and all persons who have contributed in the preparation of my project to complete it successfully. I am also thankful to who helped me a lot in my crises and gave me full support toward the completion of my project.

I would like to thank my family who gave their lasting encouragement in my studies and enduring these all expenses and supporting me in all events, so that I could be successful in my life time. I specially thank to my mother whose prayers have helped me to keep safe from every dark region of life. Special thank to my father who help me in joining this prestigious university and helped me to make my future brighter.

I am also very much grateful to all my friends and colleagues who gave their precious time to help me and giving me their ever devotion and all valuable information which I really need to complete my project.

Further I am thankful to Near East University academic staff and all those persons who helped me or encouraged me incompletion of my project. Thanks!"

ABSTRACT

Until 1976, a single key was always used to both encode the message and to decode it. Consequently, for two people to communicate securely, they must both have a copy of the same key. This raises extreme problems in transferring the key securely.

In 1976, a law called Public Key Cryptography developed with a new approach. In this approach each person has two keys, which they generate with special software at the same time. They can relate the keys but not in any way which can be computed externally. One the private key is kept secret. The other the public key can be given freely to anyone. Something encrypted with the private key can only be decrypted with the public key. Something encrypted with the public key can only be decrypted with the private key. This means that someone can send a message without getting a secret key by simply encrypting public key. This utterly changes the usefulness of cryptography previously physical couriers were needed to transport the single keys to both end of an anticipated communication path because no electronic path could be trusted with the key. Public key cryptography, when properly implemented and used, enables people to communicate with complete secrecy, and to sign documents, with all practical terms of absolute security without ever having to exchange something like a single symmetric key which must be kept secret.

TABLE OF CONTENTS

ACKNOWLEDGEMENT	i
ABSTRACT	ii
TABLE OF CONTENTS	iii
INTRODUCTION	1
1. INTRODUCTION TO CRYPTOGRAPHY	3
1.1 Overview	3
1.2 Cryptography	3
1.3 Basic Functions and Concepts	7
1.3.1 Function	7
1.3.2 Basic Terminology and Concepts	8
1.3.2.1. Encryption Domains and Co-domains	8
1.3.2.2 Encryption and Decryption Transformations	8
1.3.2.3 Achieving Confidentiality	9
1.3.2.4 Communication Participants	10
1.3.2.5. Channels	10
1.3.2.6 Security	11
1.3.2.7 Network Security in General	11
1.4 Symmetric-key Encryption	12
1.4.1 Block Ciphers	13
1.4.2 Stream Ciphers	14
1.4.3 The Key Space	14
1.5 Digital Signatures	14
1.5.1. Nomenclature and Set-up	14
1.6 Public-key Cryptography	15
1.7 Hash Functions	16
1.8 Protocols, Mechanisms	17
1.8.1 Protocol and Mechanism Failure	17
1.9 Classes of Attacks and Security Models	18
1.9.1 Attacks on Encryption Schemes	18
1.9.2 Attacks on Protocols	19
2. CRYPTOGRAPHY FUNCTIONS	20
2.1 Overview	20
2.2 Block Ciphers	20
2.2.1 Iterated Block Cipher	20
2.2.2 Electronic Codebook (ECB) Mode	21
2.2.3 Cipher Block Chaining (CBC) Mode	22
2.2.4 Feistel Ciphers	23
2.2.5 Data Encryption Standard (DES)	24
2.2.5.1 Triple DES	25
2.3 Stream Ciphers	25
2.3.1 Linear Feedback Shift Register	26

2.3.1.1 Shift Register Cascades	26
2.3.1.2 Shrinking and Self-Shrinking Generators	27
2.3.2 Other Stream Ciphers	27
2.3.2.1 One-time Pad	28
2.4 Hash Functions	28
2.4.1 Hash functions for hash table lookup	29
2.5 Attacks on Ciphers	30
2.5.1 Exhaustive Key Search	30
2.5.2 Differential Cryptanalysis	30
2.5.3 Linear Cryptanalysis	31
2.5.4 Weak Key for a Block Cipher	31
2.5.5 Algebraic Attacks	32
2.5.6 Data Compression Used With Encryption	32
2.6 When an Attack Becomes Practical	33
2.7 Strong Password-Only Authenticated Key Exchange	34
2.7.1 The Remote Password Problem	35
2.7.2 Characteristics of Strong Password-only Methods	36
2.7.2.1 SPEKE	36
2.7.2.2 DH-EKE	38
2.8 Different kinds of Security Attacks	39
2.8.1 Discrete Log Attack	39
2.8.2 Leaking Information	39
2.8.2.1 DH-EKE Partition Attack	40
2.8.2.2 SPEKE Partition Attack	40
2.8.3 Stolen Session Key Attack	41
2.8.4 Verification Stage Attacks	41
2.8.5 The "password-in-exponent" Attack	41
2.9 A Logic of Authentication	43
3. DATA ENCRYPTION STANDARD (DES)	45
3.1 Overview	45
3.2 Simplified DES(S _{DES})	48
3.2.1 Subkey generation	48
3.2.2 Relation with DES	49
3.3 History of DES	49
3.4 How DES Works in Detail	51
3.4.1 Step 1 find 16 sub keys, each of which is 48-bits long	53
3.4.2 Step 2: Encode each 64-bit block of data	58
3.4.3 DES Modes of Operation	66
3.4.4 Some Preliminary Examples of DES	66
3.5 Cracking DES	68
3.6 Triple-DES	70
4. NETWORK SECURITY	71
4.1 Overview	71
4.2 What is a Network?	71
4.3 The ISO/OSI Reference Model	71
4.4 Overview of TCP/IP	73

4.4.1 Open Design	73
4.4.2 IP	73
4.4.3 IP Address	74
4.4.3.1 Static And Dynamic Addressing	74
4.4.3.2 Attacks against	75
4.4.3.3 IP spoofing	75
4.4.4 TCP and UDP Ports	75
4.4.4.1 TCP	76
4.4.4.2 UDP	76
4.5 Risk Management: The Game of Security	76
4.5.1 Security Risks	77
4.5.2 Security Threats	78
4.6 Types and Sources of Network Threats	79
4.6.1 Denial-of-Service	79
4.6.2 Unauthorized Access	80
4.6.2.1 Executing Commands Illicitly	80
4.6.2.2 Confidentiality Breaches	81
4.6.2.3 Destructive Behavior	81
4.6.3 Where Do They Come From?	82
4.6.4 Lessons Learned	82
4.6.4.1 Hope you have Backups	82
4.6.4.2 Don't Put Data where it doesn't need to be	83
4.6.4.3 Avoid Systems with Single Points of Failure	83
4.6.4.4 Stay Current with Relevant Operating System Patches	83
4.6.4.5 Watch for Relevant Security Advisories	83
4.6.4.6 Have Someone on Staff be Familiar with Security	83
4.7 Generation and Distribution of Keys	84
4.8 Modification of Derived Key Base	85
CONCLUSION	86
REFERNCES	87

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. By connecting providers and suppliers around the world, and allowing them to interact via automated mechanisms, technology is opening amazing opportunities, mostly the result of removing barriers to communication and commerce. However, with this come risks of illegitimate, malicious use and access of information, by an adversary abusing the ease of storage, processing and communication. There are risks and threats associated with the existing commercial and social mechanisms. Such as expose of secret information from storage or communication, e.g. credit card numbers or medical records. Modification in information stored or communicated, e.g. moving funds illegitimately. Duplicating and selling copyrighted text or music and last is misrepresent herself when communicating, creating false image, and using this to cheat.

Cryptography is not a trivial area. Since its goal is to govern the use of information, preventing unauthorized use, simulations and experimentation cannot test cryptographic mechanisms. Furthermore, weaknesses are often hard to find, and often finding a weakness involves substantial innovation and ingenuity. In fact, there is a branch of cryptography, called cryptanalysis, dedicated to breaking cryptographic mechanisms and their applications. The ultimate test of any cryptographic mechanism is when a very large effort by dedicated researchers and by actual adversaries fails to find a weakness in it. However, this is rarely a useful test for new mechanisms and systems. This makes precise definitions and proofs of security extremely important.

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

In my second chapter I have explained various functions techniques used in cryptography in detail. It includes ciphers, a technique use to code data then we have hash function and the authentication methods and threats to the cryptography as how some one can break through to check the secure information

My third chapter presents the data encryption standard (DES), this chapter describes briefly simplified DES (S_DES) and how DES algorithm works in details, and the history of DES, I have explained how DES works in details, 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.

My last fourth chapter is about the network security. As cryptography is the techniques and network security is overall security of the information on the network. I have explained in detail about the network and about OSI layer model then what protocols are and how they have threat for different attacks. And I wrote about the security risks and security threats, and then I have explained about the Distribution of Keys and how they make the network security possible and explain Modification of Derived Key Base .

1. INTRODUCTION TO CRYPTOGRAPHY

1.1 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. Some of these objectives are listed in Table 1.1. 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

1.2 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.

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.

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 1.1 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.

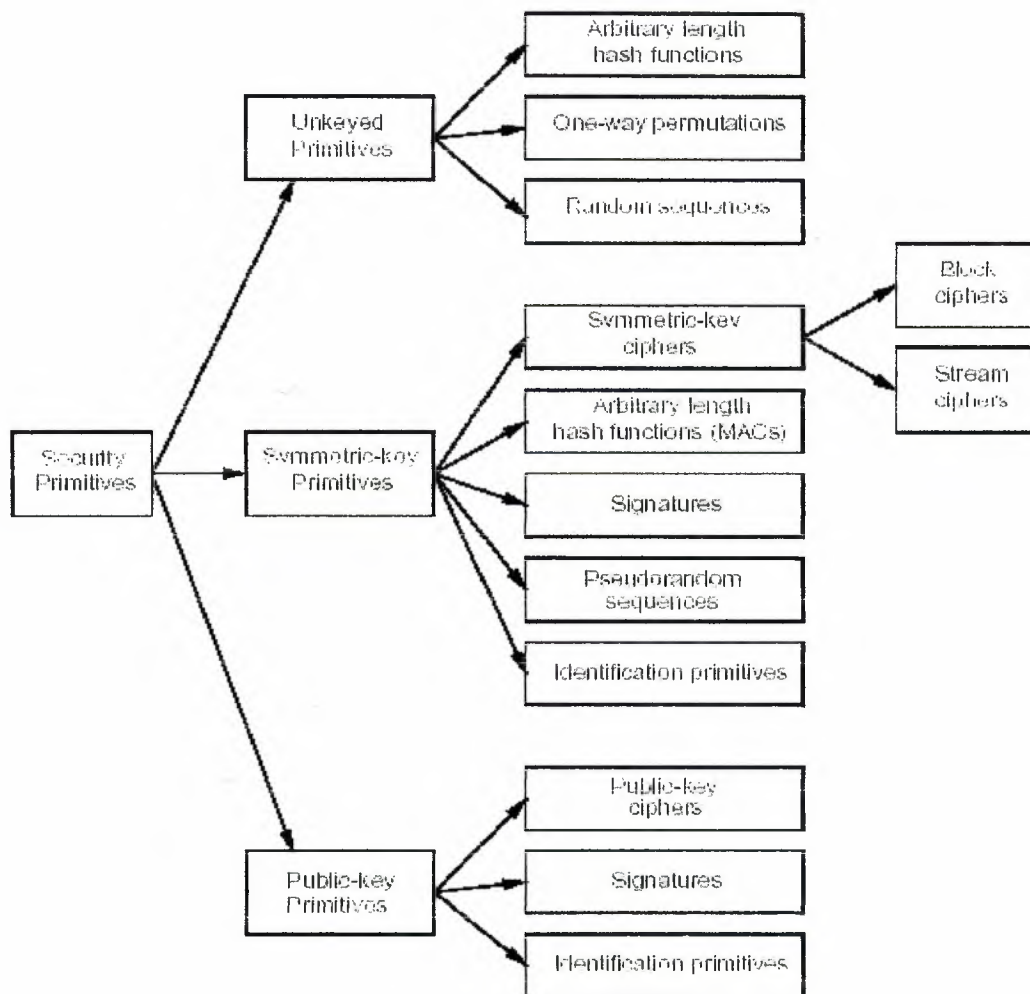


Figure 1.1 A taxonomy of cryptographic primitives.

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.

1.3 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.

1.3.1 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 $x \in X$) 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 $f: X \rightarrow Y$.

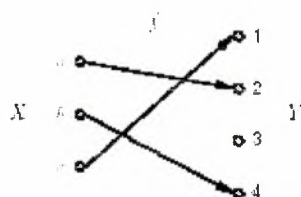


Figure 1.2 A function f from a set X to a set Y .

- **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 $f: X \rightarrow Y$ is 1-1 and $\text{Im}(f) = Y$, 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 $x \in X$ but for essentially all elements $y \in \text{Im}(f)$ it is “computationally infeasible” to find any $x \in X$ such that $f(x) = y$.
- Trapdoor one-way functions: A trapdoor one-way function is a one-way function $f: X \rightarrow Y$ 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., $p: S \rightarrow S$).
- Involutions: Involutions have the property that they are their own inverses. (i.e., $f: S \rightarrow S$).

1.3.2 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.

1.3.2.1 Encryption Domains and Co-domains

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

1.3.2.2 Encryption and Decryption Transformations

Encryption is the process of transforming information so it is unintelligible to anyone but the intended recipient. Decryption is the process of transforming encrypted information so that it is intelligible again. A cryptographic algorithm, also called a cipher, is a mathematical function used for encryption or decryption. In most cases, two related functions are employed, one for encryption and the other for decryption.

- \mathcal{K} denotes a set called the key space. An element of \mathcal{K} is called a key.
- Each element $e \in \mathcal{K}$ uniquely determines a bijection from \mathcal{M} to \mathcal{C} , denoted by \mathcal{E}_e .
- \mathcal{D}_d denotes a bijection from \mathcal{C} to \mathcal{M} and \mathcal{D}_d is called a decryption function.
- The process of applying the transformation \mathcal{E}_e to a message $m \in \mathcal{M}$ is usually referred to as encrypting m or the encryption of m .
- The process of applying the transformation \mathcal{D}_d to a ciphertext 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 $(e; d)$.

1.3.2.3 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 $m \in \mathcal{M}$ to Bob, she computes $c = \mathcal{E}_e(m)$ and transmits this to Bob. Upon receiving c , Bob computes $\mathcal{D}_d(c) = m$ 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 1.3 provides a simple model of a two-party communication using encryption.

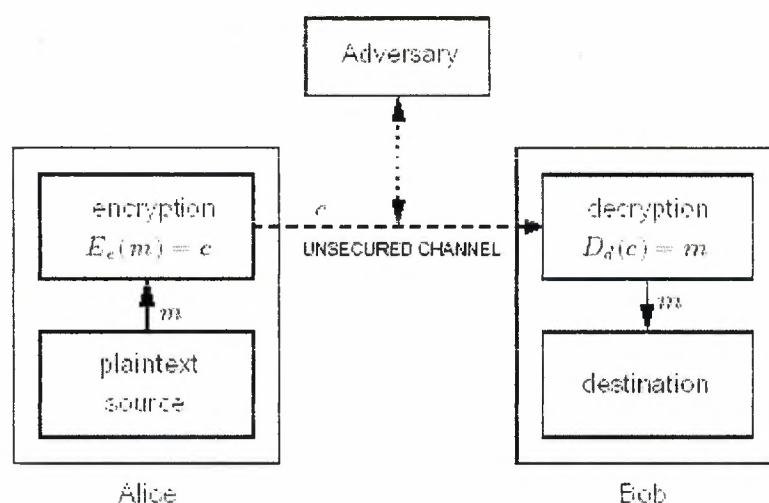


Figure 1.3 Schematic of a two-party communication.

1.3.2.4 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.

1.3.2.5. 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.

1.3.2.6 Security

A fundamental principle in cryptography is that the sets \mathcal{M} , \mathcal{C} , \mathcal{K} , $\{\mathcal{E}e: e \in \mathcal{K}\}$, $\{\mathcal{D}d: d \in \mathcal{K}\}$ 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.

1.3.2.7 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.

1.1 Symmetric-key Encryption

Consider an encryption scheme consisting of the sets of encryption and decryption transformations $\{E_e: e \in \mathcal{K}\}$ and $\{D_d: d \in \mathcal{K}\}$, respectively, where \mathcal{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 1.4, with the addition of the secure channel.

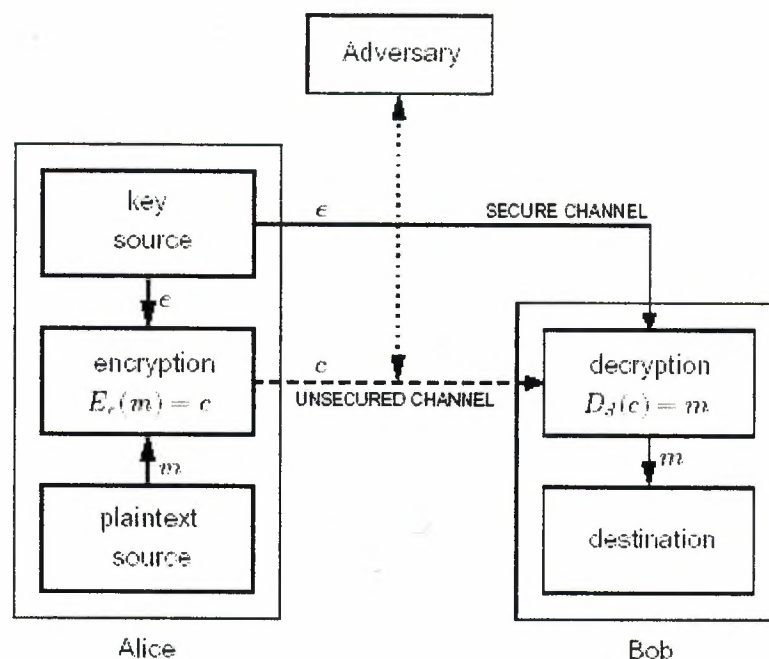


Figure 1.4 Two-party communication using encryption, with a secure channel

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.

1.4.1 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 \mathcal{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

1.4.2 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

1.4.3 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.

1.5 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.

1.5.1. Nomenclature and Set-up

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

- \mathcal{M} is the set of messages which can be signed.
- \mathcal{S} is a set of elements called signatures, possibly binary strings of a fixed length.

- S_A is a transformation from the message set \mathcal{M} to the signature set \mathcal{S} , and is called a signing transformation for entity A .
- V_A is a transformation from the set $\mathcal{M} \times \mathcal{S}$ to the set $\{\text{true}, \text{false}\}$. 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 .

1.6 Public-key Cryptography

The concept of public-key encryption is simple and elegant, but has far-reaching consequences. Let $\{E_e: e \in \mathcal{K}\}$ be a set of encryption transformations, and let $\{D_d: d \in \mathcal{K}\}$ be the set of corresponding decryption transformations, where \mathcal{K} is the key space. Consider any pair of associated encryption/decryption transformations (E_e, D_d) and suppose that each pair has the property that knowing E_e it is computationally infeasible, given a random ciphertext $c \in \mathcal{C}$, to find the message $m \in \mathcal{M}$ such that $E_e(m) = c$. This property implies that given e it is infeasible to determine the corresponding decryption key d . E_e 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.

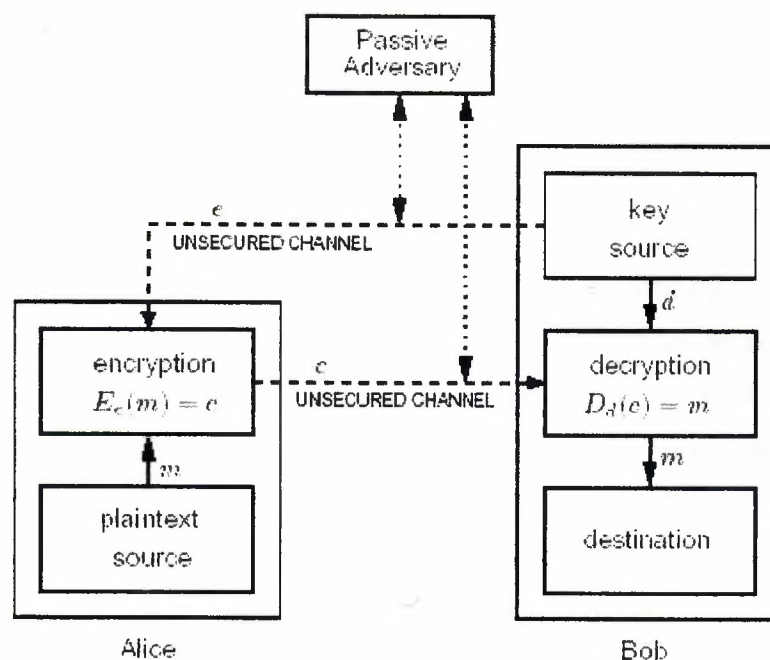


Figure 1.5 Encryption using public-key techniques.

1.7 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).

1.8 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.

1.8.1 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.

1.9 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.

1.9.1 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.

1.9.2 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.

2. CRYPTOGRAPHY FUNCTIONS

2.1 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 stream ciphers. 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.

2.2 Block Ciphers

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.

2.2.1 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 subkey. The set of subkeys 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.

2.2.2 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 ciphertext will be the same.

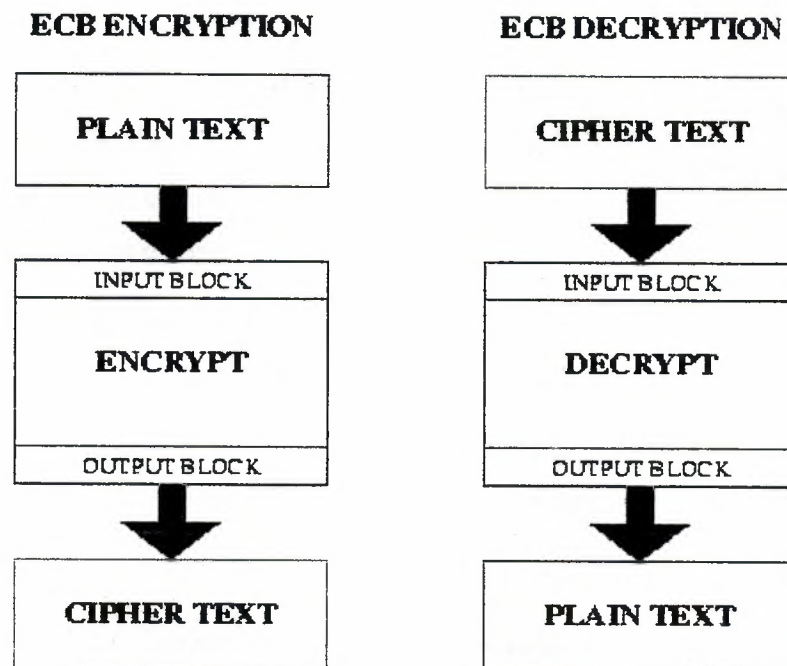


Figure 2.1: Shows a ECB Encryption/Decryption Model

2.2.3 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.

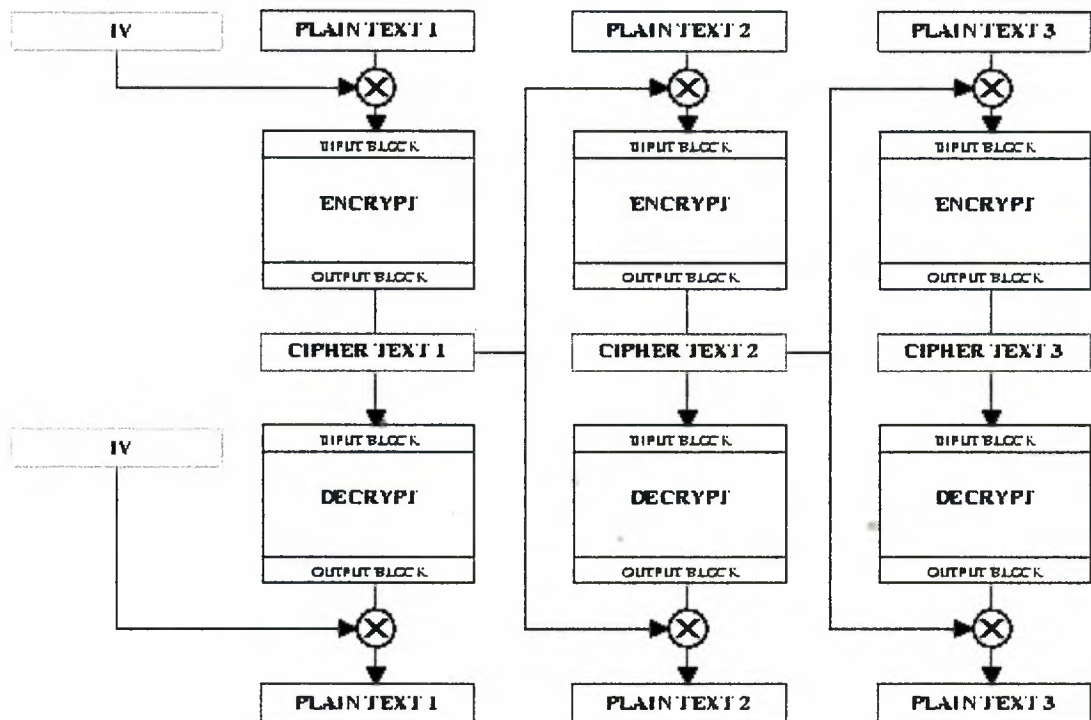


Figure 2.2: Shows a CBC Encryption/Decryption Model

2.2.4 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. Ideally, the output of the hash function should appear to be random bits from which nothing can be determined about the input(s).

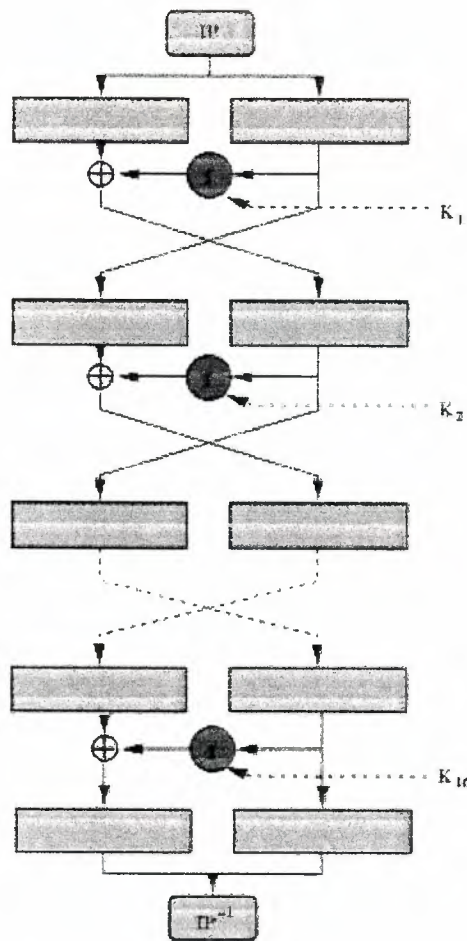


Figure 2.3: Shows a Feistel Model

2.2.5 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 subkeys, one for each cycle. To decrypt, the identical algorithm is used, but the order of subkeys 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 subkeys 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.

2.2.5.1 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.

2.3 Stream Ciphers

A stream cipher is a symmetric encryption algorithm. Stream ciphers can be designed to be exceptionally fast, much faster in fact than any block cipher. While block ciphers operate on large blocks of data, stream ciphers typically operate on smaller units of plaintext, usually bits. The encryption of any particular plaintext with a block cipher will result in the same ciphertext when the same key is used. With a stream cipher, the transformation of these smaller plaintext units will vary, depending on when they are encountered during the encryption process.

A stream cipher generates what is called a keystream and encryption is provided by combining the keystream with the plaintext, usually with the bitwise XOR operation. The generation of the keystream can be independent of the plaintext and ciphertext or it can depend on the data and its encryption.

Current stream ciphers are most commonly attributed to the appealing of theoretical properties of the one-time pad, but there have been no attempts to standardize on any particular stream cipher proposal as has been the case with block ciphers. Interestingly, certain modes of operation of a block cipher effectively transform it into a keystream

generator and in this way; any block cipher can be used as a stream cipher. However, stream ciphers with a dedicated design are likely to be much faster.

2.3.1 Linear Feedback Shift Register

A Linear Feedback Shift Register (LFSR) is a mechanism for generating a sequence of binary bits. The register consists of a series of cells that are set by an initialization vector that is, most often, the secret key. The behavior of the register is regulated by a clock and at each clocking instant, the contents of the cells of the register are shifted right by one position, and the XOR of a subset of the cell contents is placed in the leftmost cell. One bit of output is usually derived during this update procedure.

LFSRs are fast and easy to implement in both hardware and software. With a sensible choice of feedback taps the sequences that are generated can have a good statistical appearance. However, the sequences generated by single LFSRs are not secure because a powerful mathematical framework has been developed over the years which allows for their straightforward analysis. However, LFSRs are useful as building blocks in more secure systems.

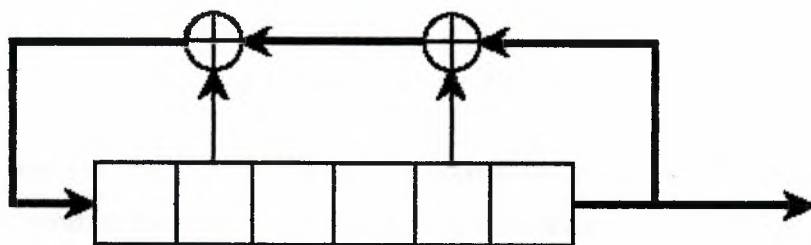


Figure 2.1: Shows a Linear Feed Back Register Model

2.3.1.1 Shift Register Cascades

A shift register cascade is a set of LFSRs connected together in such a way that the behavior of one particular LFSR depends on the behavior of the previous LFSRs in the cascade. This dependent behavior is usually achieved by using one LFSR to control the clock of the following LFSR. For instance one register might be advanced by one step if the

preceding register output is 1 and advanced by two steps otherwise. Many different configurations are possible and certain parameter choices appear to offer very good security.

2.3.1.2 Shrinking and Self-Shrinking Generators

It is a stream cipher based on the simple interaction between the outputs from two LFSRs. The bits of one output are used to determine whether the corresponding bits of the second output will be used as part of the overall keystream. The shrinking generator is simple and scaleable, and has good security properties. One drawback of the shrinking generator is that the output rate of the keystream will not be constant unless precautions are taken. A variant of the shrinking generator is the self-shrinking generator, where instead of using one output from one LFSR to “shrink” the output of another, the output of a single LFSR is used to extract bits from the same output.

2.3.2 Other Stream Ciphers

There are a vast number of alternative stream ciphers that have been proposed in cryptographic literature as well as an equally vast number that appear in implementations and products world-wide. Many are based on the use of LFSRs since such ciphers tend to be more amenable to analysis and it is easier to assess the security that they offer.

There are essentially four distinct approaches to stream cipher design. The first is termed the information-theoretic approach explained in one-time pad. The second approach is that of system-theoretic design. In essence, the cryptographer designs the cipher along established guidelines which ensure that the cipher is resistant to all known attacks. While there is, of course, no substantial guarantee that future cryptanalysis will be unsuccessful, it is this design approach that is perhaps the most common in cipher design. The third approach is to attempt to relate the difficulty of breaking the stream cipher to solving some difficult problem. This complexity-theoretic approach is very appealing, but in practice the ciphers that have been developed tend to be rather slow and impractical. The final approach is that of designing a randomized cipher. Here the aim is to ensure that the cipher is

resistant to any practical amount of cryptanalytic work rather than being secure against an unlimited amount of work.

2.3.2.1 One-time Pad

A one-time pad, sometimes called the Vernam cipher, uses a string of bits that is generated completely at random. The keystream is the same length as the plaintext message and the random string is combined using bitwise XOR with the plaintext to produce the ciphertext. Since the entire keystream is random, an opponent with infinite computational resources can only guess the plaintext if he sees the ciphertext. Such a cipher is said to offer perfect secrecy and the analysis of the one-time pad is seen as one of the cornerstones of modern cryptography.

2.4 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.

2.4.1 Hash functions for hash table lookup

A hash function for hash table lookup should be fast, and it should cause as few collisions as possible. If you know the keys you will be hashing before you choose the hash function, it is possible to get zero collisions -- this is called perfect hashing. Otherwise, the best you can do is to map an equal number of keys to each possible hash value and make sure that similar keys are not unusually likely to map to the same value. Unfortunately, that

hash is only average. The problem is the per-character mixing: it only rotates bits, it doesn't really mix them. Every input bit affects only 1 bit of hash until the final %. If two input bits land on the same hash bit, they cancel each other out. Also, % can be extremely slow.

2.5 Attacks on Ciphers

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


2.5.1 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 modern 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.

2.5.2 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.

2.5.3 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.

2.5.4 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.

2.5.5 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.

2.5.6 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.

2.6 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.

2.7 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.

2.7.1 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 plaintext 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.

2.7.2 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

(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.

2.7.2.1 SPEKE

The simple password exponential key exchange (SPEKE) has two stages. The first stage uses a DH exchange to establish a shared key K , but instead of the commonly used fixed primitive base g , a function f converts the password S into a base for exponentiation. The rest of the first stage is pure Diffie-Hellman, where Alice and Bob start out by choosing two random numbers R_A and R_B :

Table 2.1: Shows First Stages of SPEKE

S1.	Alice computes:	$Q_A = f(S)^{R_A} \bmod p,$	$A \rightarrow B: Q_A.$
S2.	Bob computes:	$Q_B = f(S)^{R_B} \bmod p,$	$B \rightarrow A: Q_B.$
S3.	Alice computes:	$K = h(Q_B^{R_A} \bmod p)$	
S4.	Bob computes:	$K = h(Q_A^{R_B} \bmod p)$	

In the second stage of SPEKE, both Alice and Bob confirm each other's knowledge of K before proceeding to use it as a session key. One way is:

Table 2.2: Shows Second Stage of SPEKE

S5.	Alice chooses random $C_A,$	$A \rightarrow B: E_K(C_A).$
S6.	Bob chooses random $C_B,$	$B \rightarrow A: E_K(C_B, C_A).$
S7.	Alice verifies that C_A is correct,	$A \rightarrow B: E_K(C_B).$
S8.	Bob verifies that C_B is correct.	

To prevent discrete log computations, which can result in the attacks the value of p^{-1} must have a large prime factor q . The function f is chosen in SPEKE to create a base of large prime order. This is different than the commonly used primitive base for DH. The use of a prime-order group may also be of theoretical importance.

Other variations of the verification stage are possible. This stage is identical to that of the verification stage of DH-EKE. More generally, verification of K can use any classical method, since K is cryptographically large. This example repeatedly uses a one-way hash function:

Table 2.3: Shows Verification Stage of SPEKE

S5.	Alice sends proof of K:	$A \rightarrow B: h(h(K))$
S6.	Bob verifies $h(h(K))$ is correct,	$B \rightarrow A: h(K)$
S7.	Alice verifies $h(K)$ is correct.	

This approach uses K in place of explicit random numbers, which is possible since K was built with random information from both sides.

2.7.2.2 DH-EKE

DH-EKE (Diffie-Hellman Encrypted Key Exchange) are the simplest of a number of methods. The method can also be divided into two stages. The first stage uses a DH exchange to establish a shared key K , where one or both parties encrypts the exponential using the password S . With knowledge of S , they can each decrypt the other's message using E_S^{-1} and compute the same key K .

Table 2.4: Shows First Stage of DH-EKE

D1.	Alice computes:	$Q_A = g^R_A \text{ mod } p,$	$A \rightarrow B: E_S(Q_A).$
D2.	Bob computes:	$Q_B = g^R_B \text{ mod } p,$	$B \rightarrow A: E_S(Q_B).$
D3.	Alice computes:	$K = h(Q_B^R_A \text{ mod } p)$	
D4.	Bob computes:	$K = h(Q_A^R_B \text{ mod } p)$	

It is widely suggested that at least one of the encryption steps can be omitted, but this may leave the method open to various types of attacks. The values of p and g , and the symmetric encryption function E_S must be chosen carefully to preserve the security of DH-

EKE. In the second stage of DH-EKE, both Alice and Bob confirm each other's knowledge of K before proceeding to use it as a session key. However, with DH-EKE the order of the verification messages can also be significant.

2.8 Different kinds of Security Attacks

Here different kinds of attacks on the security in authentication which have been observed so far and which are expected are explained in detail.

2.8.1 Discrete Log Attack

As the security of these schemes rests primarily on exponentiation being a one-way function, there is a general threat of an attacker computing the discrete logarithms on the exponentials. Known methods of discrete log require a massive pre-computation for each specific modulus. Modulus size is a primary concern. No method is currently known that could ever compute the discrete log for a safe modulus greater than a couple thousand bits; however a concerted attack on a 512 bit modulus may be soon feasible with considerable expense. Somewhere in between is an ideal size balancing speed against the need for security, in a given application.

It is noted that if we assume that a discrete log pre-computation has been made for the modulus, a password attack must also compute the specific log for each entry in the password dictionary (until a match is found). It is also noted that for any session established with a modulus vulnerable to log attack, perfect forward secrecy is no longer guaranteed, providing another reason for keeping the discrete log computation out of reach. The feasibility of a pre-computed log table remains a primary concern, and the efficiency of the second phase of the attack is secondary.

2.8.2 Leaking Information

If one is not careful, the exchanged messages Q_x may reveal discernible structure, and can "leak" information about S , enabling a partition attack. This section shows how to prevent these attacks.

2.8.2.1 DH-EKE Partition Attack

In DH-EKE, Alice and Bob use a Diffie-Hellman exponential key exchange in the group Z_p^* , with a huge prime p , where $p-1$ has a huge prime factor q . Then we use the traditional preference for g as a primitive root of p . In fact, g must be primitive to prevent a partition attack by an observer. A third party can do trial decryptions of $E_S(g^R \bmod p)$ using a dictionary of S_i . If g is not primitive, a bad guess S_i is confirmed by a primitive result. In general, the encrypted exponentials Q_x must contain no predictable structure to prevent this attack against DH-EKE. Constraining g to be primitive insures a random distribution across Z_p^* .

2.8.2.2 SPEKE Partition Attack

Using a primitive base is not required in SPEKE. If the base $f(S)$ is an arbitrary member of Z_p^* , since the exponentials are not encrypted, an observer can test the result for membership in smaller subgroups. When the result is a primitive root of p , he knows that the base also is primitive. For a safe prime p , this case reveals 1 bit of information about S . When p varies, as has been recommended when using a reduced modulus size, new information from runs with different p allow a partition attack to reduce a dictionary of possible S_i . When, for any S , the base $f(S)$ is a generator of a particular large prime subgroup, and then no information is leaked through the exponential result. Suitable functions for $f(S)$ create a result of known large order. We assume the use of a large prime-order base in SPEKE for the rest of the discussion. Because SPEKE does not encrypt the exponentials, a formal analysis of security may be simpler to achieve for SPEKE than for DH-EKE. The prime-order subgroup is the same as that used in the DSA and Digital signature methods.

2.8.3 Stolen Session Key Attack

In an analysis of several flavors of EKE, where a stolen session key K is used to mount a dictionary attack on the password. The attack on the public-key flavor of EKE is also noted which correctly points out that DH-EKE resists this attack (as does SPEKE). Resistance to this attack is closely related to perfect forward secrecy, which also isolates one kind of sensitive data from threats to another. We note that, in DH-EKE, a stolen value of R_A in addition to K permits a dictionary attack against the password S . For each trial password S_i , the attacker computes:

$$K' = (E_{S_i}^{-1}(E_S(g_B^R)))^{R_A}$$

When K' equals K , he knows that S_i equals S . SPEKE is also vulnerable to an attack using R_A to find S . These concerns highlight the need to promptly destroy ephemeral sensitive data, such as R_A and R_B . It also notes a threat when the long-term session key K is used in an extra stage of authentication of the extended A-EKE method; a dictionary attack is possible using the extra messages. To counter this threat, one can use K for the extra stage, set $K' = h(K)$ using a strong one-way function, and promptly discard K .

2.8.4 Verification Stage Attacks

The verification stage of either DH-EKE or SPEKE is where both parties prove to each other knowledge of the shared key K . Because K is cryptographically large, the second stage is presumed to be immune to brute-force attack, and thus verifying K can be done by traditional means. However, the order of verification may be important to resist the protocol attack against DH-EKE.

2.8.5 The "password-in-exponent" Attack

It is generally a good idea for $f(S)$ to create a result of the same known order for all S , so that testing the order of the exponential doesn't reveal information about S . When considering suitable functions, it may be tempting to choose $f(S) = g_c^{h(S)}$ for some fixed prime-order g_c and some well-known hash function h . Unfortunately, while this is a convenient way to convert an arbitrary number into a generator of a prime-order group, it

creates an opening for attack. To show the attack, let's assume that $g_c = 2$, and $h(S) = S$, so that $f(S) = 2^S$. Alice's protocol can be rewritten as:

1. Choose a random R_A .
2. Compute $Q_A = 2^{(S R_A)} \bmod p$.
3. Send Q_A to Bob.
4. Receive Q_B from Bob.
5. Compute $K = Q_B^{R_A} \bmod p$.

Bob should perform his part, sending Q_B to Alice. The problem is that an attacker Barry can perform a dictionary attack off-line after performing a single failed exchange. His initial steps are:

1. Choose a random X .
2. Compute $Q_B = 2^X$.
3. Receive Q_A from Alice
4. Send Q_B to Alice.
5. Receive verification data for K from Alice.

Barry then goes off-line to perform the attack as follows:

For each candidate password S' :

Compute $K' = (Q_B^X)^{1/S'} \bmod p$.

Compare Alice's verification message for K to K' , when they match he knows that $S' = S$.

This attack works because:

$$\begin{aligned}
 K' &= Q_A^{(X/S')} \bmod p \\
 &= 2^{(S R_A)(X/S')} \bmod p \\
 &= 2^{(X R_A S/S')} \bmod p
 \end{aligned}$$

$$= Q_B^{(R_A^{S/S'})} \bmod p$$

$$= K^{(S/S')} \bmod p$$

Thus, when $S' = S$, $K' = K$. More generally, the attack works because the dictionary of passwords $\{S_1, S_2 \dots S_n\}$ is equivalent to a dictionary of exponents $E = \{e_1, e_2 \dots e_n\}$, such that for a given fixed generator g_c , the value of $f(S_i)$ for each candidate can be computed as $g_c^{e_i}$. This allows the password to be effectively removed from the DH computation.

In general, we must insure that no such dictionary E is available to an attacker. We should note that while it is true that for any function f there will always be some fixed g_c and hypothetical dictionary E that corresponds to $f(S)$, for most functions f , computing the value of each e_i requires a discrete log computation. This makes the dictionary E generally unknowable to anyone. As a specific example, for the function $f(S) = S$, the attack is infeasible. The password-in-exponent attack is possible only when $f(S)$ is equivalent to exponentiation (within the group) of some fixed g_c to a power which is a known function of S .

2.9 A Logic of Authentication

In computer networks the communicating parties share not only the media, but also the set of rules on how to communicate. These rules, or protocols, have become more and more important in communication networks and distributed computing. However, the increase of the knowledge of the communication protocols has also brought up the question of how to secure the communication against intruders. To solve this, a large number of cryptographic protocols have been produced.

Cryptographic protocols were developed to combat against various attacks of intruders in computer networks. Nowadays, the comprehension is that the security of data should rely on the underlying cryptographic technology, and that the protocols should be open and available. However, many protocols have been found to be vulnerable to attacks that do not require breaking the encryption, but instead manipulate the messages in the

protocol to gain some advantage. The advantages range from the compromise of confidentiality to the ability to impersonate another user.

As there are different protocol designs decisions appropriate to different circumstances, there also exists a variety of authentication protocols. Protocols often differ in their final states, and sometimes they even depend on assumptions that one would not care to make. To understand what is really accomplished with such a protocol, a formal description method is needed. The goal of the logic of authentication is to formally describe the knowledge and the beliefs of the parties involved in authentication, the evolution of the knowledge and the beliefs while analyzing the protocol step by step. After the analysis, all the final states of the protocol are set out.

3. DATA ENCRYPTION STANDARD (DES)

3.1 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).

3.2 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).

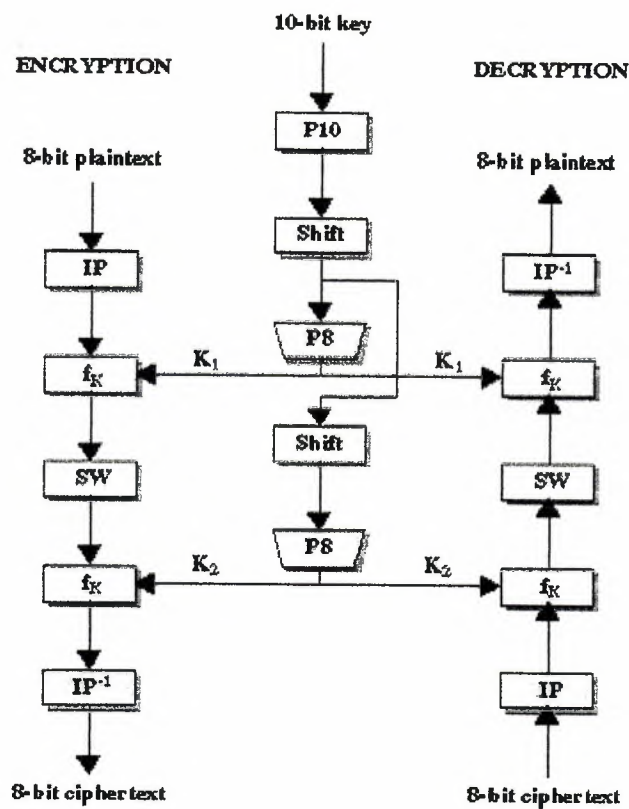


Figure 3.1 S_DES scheme

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

$f_K(L, R) = L \text{ XOR } F(R, K_i)$, 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

$n_1 n_2 n_3 n_4$ then $S_i[n_1 n_4][n_2 n_3]$

Example:

$R = 1010$

$E/P \ 0101 \ 0101$

$K_1 = 1010\ 0100$

XOR $1111\ 0001$

$S_0[11][11] = 10$ $S_1[01][00] = 10 \rightarrow P_4 = 0011$

3.2.1 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 :

$K_1 = P_8(LS(P_{10}(key)))$

$K_2 = P_8(LS(LS(P_{10}(key))))$

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

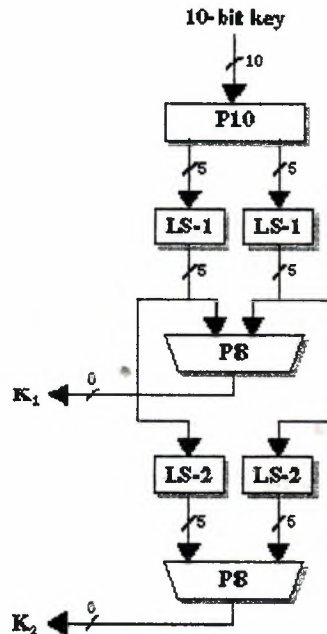


Figure 3.2 key Generation of S_DES

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

3.2.2 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.

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

3.3 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.

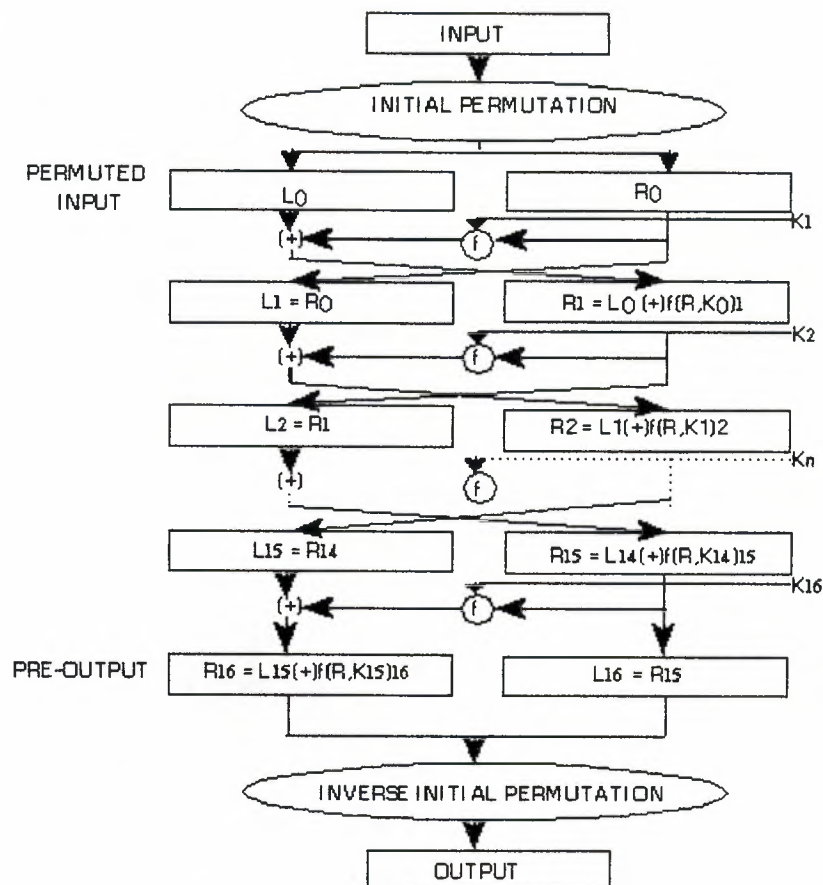
NBS waited for the responses to come in. It received none until August 6, 1974, three days before Nixon's resignation, when IBM submitted a candidate that it had developed internally under the name LUCIFER. After evaluating the algorithm with the help of the National Security Agency (NSA), the NBS adopted 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. $K = 00010011\ 00110100\ 01010111\ 01111001\ 10011011\ 10111100\ 11011111\ 11110001$.

3.4 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.)



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:

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

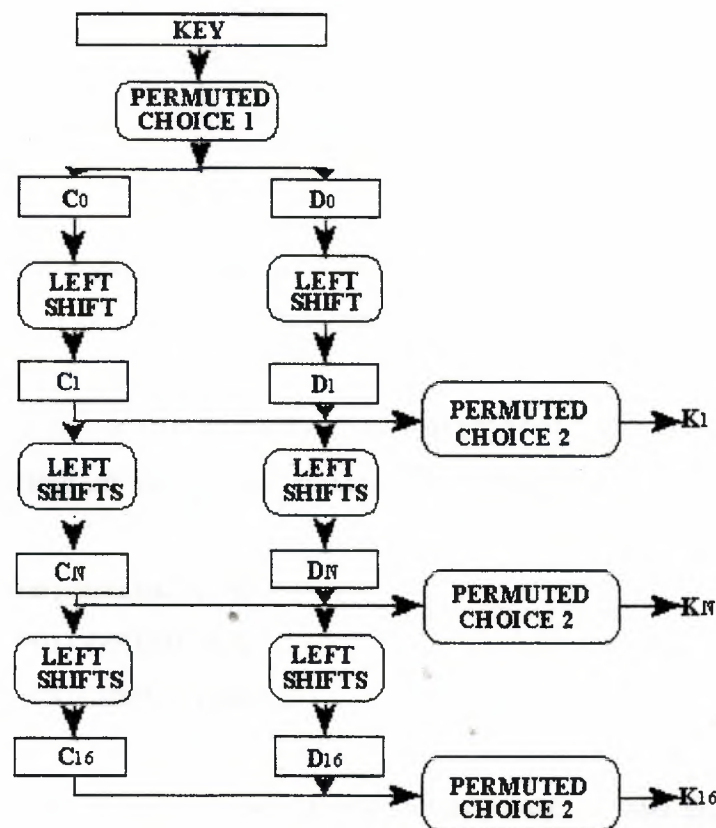


Figure 3.4 DES Key Setup

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 3.1 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 , $1 \leq n \leq 16$. Each pair of blocks C_n and D_n is formed from the previous pair C_{n-1} and D_{n-1} , respectively, for $n = 1, 2, \dots, 16$, 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 3.2

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 $1 \leq n \leq 16$, by applying the following permutation table to each of the concatenated pairs $C_n D_n$. Each pair has 56 bits, but PC-2 only uses 48 of these.

Table 3.3 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 $C_n D_n$, the second bit the 17th, and so on, ending with the 48th bit of K_n being the 32th bit of $C_n D_n$.

Example: For the first key we have $C_1 D_1 = 1110000 1100110 0101010 1011111 1010101 0110011 0011110 0011110$

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

$K_2 = 011110 \quad 011010 \quad 111011 \quad 011001 \quad 110110 \quad 111100 \quad 100111 \quad 100101$
 $K_3 = 010101 \quad 011111 \quad 110010 \quad 001010 \quad 010000 \quad 101100 \quad 111110 \quad 011001$
 $K_4 = 011100 \quad 101010 \quad 110111 \quad 010110 \quad 110110 \quad 110011 \quad 010100 \quad 011101$

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.

3.4.2 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.

Table 3.4 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 $1 \leq n \leq 16$, 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 3.5 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

$$\begin{aligned} K_1 &= 000110 \quad 110000 \quad 001011 \quad 101111 \quad 111111 \quad 000111 \quad 000001 \quad 110010 \\ E(R_0) &= 011110 \quad 100001 \quad 010101 \quad 010101 \quad 011110 \quad 100001 \quad 010101 \quad 010101 \\ K_1 + E(R_0) &= 011000 \quad 010001 \quad 011110 \quad 111010 \quad 100001 \quad 100110 \quad 010100 \quad 100111. \end{aligned}$$

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 $S_i(B_i)$ refers to the output of the i -th S box.

To repeat, each of the functions S_1, S_2, \dots, S_8 , takes a 6-bit block as input and yields a 4-bit block as output. The table to determine S_i is shown and explained in table 5.6:

Table 3.6 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

If S_I is the function defined in this table and B is a block of 6 bits, then $S_I(B)$ 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 $S_I(B)$ of S_I 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 $S_I(011011) = 0101$.

The tables defining the functions S_1, \dots, S_8 are the following in table 5.7:

Table 3.7

S1

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

$$f = P(S_1(B_1)S_2(B_2)...S_8(B_8))$$

The permutation P is defined in the table 5.8. P yields a 32-bit output from a 32-bit input by permuting the bits of the input block.

Table 3.8 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

$$f = 0010\ 0011\ 0100\ 1010\ 1010\ 1001\ 1011\ 1011$$

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

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

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 table 5.9:

Table 3.9 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.

3.4.3 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

3.4.4 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" which are also *apparently* 16 hexadecimal numbers long or *apparently* 64 bits long. However, every 8th key bit is ignored in the DES algorithm, so that the effective key size is 56 bits. But, in any case, 64 bits (16 hexadecimal digits) is the round number upon which DES is organized.

For example, if we take the plaintext message "8787878787878787", and encrypt it with the DES key "0E329232EA6D0D73", we end up with the 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!)

3.5 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).

3.6 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} .)

4. NETWORK SECURITY

4.1 Overview

A basic understanding of computer networks is requisite in order to understand the principles of network security. In this section, we'll cover some of the foundations of computer networking, also we'll cover some of the threats and the risks that managers and administrators of computer networks need to confront, and then some tools that can be used to reduce the exposure to the risks of network computing. Once we've covered this, we'll go back and cover the process of protecting data and equipment from unauthorized access. And we'll include a brief description of network security concepts and technology.

4.2 What is a Network?

A set of interlinking lines resembling a net, a network of roads || an interconnected system, a network of alliances." This definition suits our purpose well: a computer network is simply a system of interconnected computers. How they're connected is irrelevant, and as we'll soon see, there are a number of ways to do this.

4.3 The ISO/OSI Reference Model

The International Standards Organization (ISO) Open Systems Interconnect (OSI) Reference Model defines seven layers of communications types, and the interfaces among them. See Figure 4.1. Each layer depends on the services provided by the layer below it, all the way down to the physical network hardware, such as the computer's network interface card, and the wires that connect the cards together.

An easy way to look at this is to compare this model with something we use daily: the telephone. In order for you and me to talk when we're out of earshot, we need a device like

a telephone. (In the ISO/OSI model, this is at the application layer.) The telephones, of course, are useless unless they have the ability to translate the sound into electronic pulses that can be transferred over wire and back again. (These functions are provided in layers below the application layer.) Finally, we get down to the physical connection: both must be plugged into an outlet that is connected to a switch that's part of the telephone system's network of switches.

If person A places a call to person B, person A picks up the receiver, and dials person B's number. This number specifies which central office to which to send my request, and then which phone from that central office to ring. Once person B answers the phone, they begin talking, and their session has begun. Conceptually, computer networks function exactly the same way.

It isn't important to memorize the ISO/OSI Reference Model's layers; but it is useful to know that they exist, and that each layer can not work without the services provided by the layer below it.

LAYER7	Application
LAYER6	Presentation
LAYER5	Session
LAYER4	Transport
LAYER3	Network
LAYER2	Data Link
LAYER1	Physical

Figure 4.1: the ISO/OSI Reference Model

4.4 Overview of TCP/IP

TCP/IP (Transport Control Protocol/Internet Protocol) is the language of the Internet. Anything that can learn to speak TCP/IP can play on the Internet. This is functionality that occurs at the Network (IP) and Transport (TCP) layers in the ISO/OSI Reference Model. Consequently, a host that has TCP/IP functionality (such as Unix, OS/2, MacOS, or Windows NT) can easily support applications (such as Netscape's Navigator) that uses the network.

TCP/IP protocols are not used only on the Internet. They are also widely used to build private networks, called internets, that may or may not be connected to the global Internet. An internet that is used exclusively by one organization is sometimes called an **intranet**

4.4.1 Open Design

One of the most important features of TCP/IP isn't a technological one: The protocol is an open protocol, and anyone who wishes to implement it may do so freely. Engineers and scientists from all over the world participate in the *IETF* (Internet Engineering Task Force) working groups that design the protocols that make the Internet work. Their time is typically donated by their companies, and the result is work that benefits everyone.

4.4.2 IP

IP is a “network layer” protocol. This is the layer that allows the hosts to actually talk to each other. Such things as carrying datagram's, mapping the Internet address to a physical network address, and routing, which takes care of making sure that all of the devices that have Internet connectivity can find the way to each other.

4.4.3 IP Address

IP addresses are analogous to telephone numbers – when you want to call someone on the telephone, you must first know their telephone number. Similarly, when a computer on the Internet needs to send data to another computer, it must first know its IP address. IP addresses are typically shown as four numbers separated by decimal points, or “dots”. For example, 10.24.254.3 and 192.168.62.231 are IP addresses.

If you need to make a telephone call but you only know the person’s name, you can look them up in the telephone directory (or call directory services) to get their telephone number. On the Internet, that directory is called the Domain Name System or DNS for short. If you know the name of a server, say www.cert.org, and you type this into your web browser, your computer will then go ask its DNS server what the numeric IP address is that is associated with that name.

4.4.3.1 Static And Dynamic Addressing

Static IP addressing occurs when an ISP permanently assigns one or more IP addresses for each user. These addresses do not change over time. However, if a static address is assigned but not in use, it is effectively wasted. Since ISPs have a limited number of addresses allocated to them, they sometimes need to make more efficient use of their addresses.

Dynamic IP addressing allows the ISP to efficiently utilize their address space. Using dynamic IP addressing, the IP addresses of individual user computers may change over time. If a dynamic address is not in use, it can be automatically reassigned to another computer as needed.

4.4.3.2 Attacks Against IP

A number of attacks against IP are possible. Typically, these exploit the fact that IP does not perform a robust mechanism for *authentication*, which is proving that a packet came from where it claims it did. A packet simply claims to originate from a given address, and there isn't a way to be sure that the host that sent the packet is telling the truth. This isn't necessarily a weakness, *per se*, but it is an important point, because it means authentication has to be provided at a higher layer on the ISO/OSI Reference Model. Today, applications that require strong host authentication (such as cryptographic applications) do this at the application layer.

4.4.3.3 IP Spoofing

This is where one host claims to have the IP address of another. Since many systems (such as router access control lists) define which packets may and which packets may not pass based on the sender's IP address, this is a useful technique to an attacker: he can send packets to a host, perhaps causing it to take some sort of action.

4.4.4 TCP and UDP Ports

TCP (Transmission Control Protocol) and UDP (User Datagram Protocol) are both protocols that use IP. Whereas IP allows two computers to talk to each other across the Internet, TCP and UDP allow individual applications (also known as "services") on those computers to talk to each other.

In the same way that a telephone number or physical mail box might be associated with more than one person, a computer might have multiple applications (e.g. email, file services, web services) running on the same IP address. Ports allow a computer to differentiate services such as email data from web data. A port is simply a number associated with each application that uniquely identifies that service on that computer. Both TCP and UDP use ports to identify services. Some common port numbers are 80 for web (HTTP), 25 for email (SMTP), and 53 for Domain Name System (DNS).

4.4.4.1 TCP

TCP is a transport-layer protocol. It needs to sit on top of a network-layer protocol, and was designed to ride atop IP. (Just as IP was designed to carry, among other things, TCP packets.) Because TCP and IP were designed together and wherever you have one, you typically have the other, the entire suite of Internet protocols are known collectively as TCP/IP.

4.4.4.2 UDP

UDP (User Datagram Protocol) is a simple transport-layer protocol. It does not provide the same features as TCP, and is thus considered “unreliable”. Again, although this is unsuitable for some applications, it does have much more applicability in other applications than the more reliable and robust TCP.

4.5 Risk Management: The Game of Security

It's very important to understand that in security, one simply cannot say “what's the best firewall?” There are two extremes: absolute security and absolute access. The closest we can get to an absolutely secure machine is one unplugged from the network, power supply, locked in a safe, and thrown at the bottom of the ocean. Unfortunately, it isn't terribly useful in this state. A machine with absolute access is extremely convenient to use: it's simply there, and will do whatever you tell it, without questions, authorization, passwords, or any other mechanism. Unfortunately, this isn't terribly practical, either: the Internet is a bad neighborhood now, and it isn't long before some bonehead will tell the computer to do something like self-destruct, after which, it isn't terribly useful to you.

This is no different from our daily lives. We constantly make decisions about what risks we're willing to accept. When we get in a car and drive to work, there's a certain risk that we're taking. It's possible that something completely out of control will cause us to become part of an accident on the highway. When we get on an airplane, we're accepting the level of risk involved as the price of convenience. However, most people have a mental picture of what an acceptable risk is, and won't go beyond that in most circumstances. If I

happen to be upstairs at home, and want to leave for work, I'm not going to jump out the window. Yes, it would be more convenient, but the risk of injury outweighs the advantage of convenience.

Every organization needs to decide for itself where between the two extremes of total security and total access they need to be. A policy needs to articulate this, and then define how that will be enforced with practices and such. Everything that is done in the name of security, then, must enforce that policy uniformly.

4.5.1 Security Risks

The first step to understanding security is to know what the potential risks are, or more specifically, to determine the type and level of security risks for the company. Security risks are unique to each organization because they are dependent on the nature of the business and the environment in which the company operates. For example, the security risks for a high profile dot com company that solely operates on the Internet will be very different from a small manufacturing company that does little on the Web.

Security risk is determined by identifying the assets that need to be protected. The assets could include customer credit card information, proprietary product formulas, employee data, the company's Web site, or other assets that are deemed to be important to the organization. Once the assets are identified, the next step is to determine the criticality of the assets to the company. For example, if the asset is considered to be very important to the company, then the level of security for that asset should be high.

The next step is assessing the likelihood of a potential attack. While security measures must always be put in place to protect the assets of the company, the risks increase as the probability of an attack rises. For example, it is more likely for an outside intruder to attempt to break into a Web site selling consumer goods than a small manufacturing company making rubber bands. Therefore, while both companies must have security measures, the company with the Web site must

deploy a higher level of security. Now that the process of determining security risk has been defined, some of the more common security risks are briefly discussed below.

4.5.2 Security Threats

The first step in evaluating security risks is to determine the threats to system security. Although the term network security has been commonly categorized as protecting data and system resources from infiltration by third-party invaders, most security breeches are initiated by personnel inside the organization. Organizations will spend hundreds of thousands of dollars on securing sensitive data from outside attack while taking little or no action to prevent access to the same data from unauthorized personnel within the organization.

The threat from hackers has been largely overstated. Individuals who fit into this group have more of a Robin Hood mentality than a destructive mentality. Most hackers, or crackers as they prefer to be called, are more interested in the thrill of breaking into the system than they are in causing damage once they succeed in gaining access. Unfortunately, there is an increasing trend for hackers to be employed by other entities as an instrument to gain access to systems.

As the amount of critical data stored on networked systems has increased, the appeal of gaining access to competitors' systems has also increased. In highly competitive industry segments, an entire underground market exists in the buying and trading of product and sales data. By gaining access to research and development information from a competitor, millions of dollars and years of research can be eliminated.

Another external threat is that of government intrusion, both from the domestic government and from foreign governments. Agencies such as the Federal Bureau of Investigation and the Internal Revenue Service can have vested interests in gaining access to critical tax and related information. Foreign governments are

especially interested in information that could represent an economic or national defense advantage

4.6 Types and Sources of Network Threats

Now, we've covered enough background information on networking that we can actually get into the security aspects of all of this. First of all, we'll get into the types of threats there are against networked computers, and then some things that can be done to protect you against various threats.

4.6.1 Denial-of-Service

DoS (Denial-of-Service) attacks are probably the nastiest, and most difficult to address. These are the nastiest, because they're very easy to launch, difficult (sometimes impossible) to track, and it isn't easy to refuse the requests of the attacker, without also refusing legitimate requests for service.

The premise of a DoS attack is simple: send more requests to the machine than it can handle. There are toolkits available in the underground community that make this a simple matter of running a program and telling it which host to blast with requests. The attacker's program simply makes a connection on some service port, perhaps forging the packet's header information that says where the packet came from, and then dropping the connection. If the host is able to answer 20 requests per second, and the attacker is sending 50 per second, obviously the host will be unable to service all of the attacker's requests, much less any legitimate requests (hits on the web site running there, for example).

- Such attacks were fairly common in late 1996 and early 1997, but are now becoming less popular
- Some things that can be done to reduce the risk of being stung by a denial of service attack include

- Not running your visible-to-the-world servers at a level too close to capacity using packet filtering to prevent obviously forged packets from entering into your network address space.
- Obviously forged packets would include those that claim to come from your own hosts, addresses reserved for private networks as defined in RFC 1918 [4], and the loop back network (127.0.0.0).
- Keeping up-to-date on security-related patches for your hosts' operating systems.

4.6.2 Unauthorized Access

“Unauthorized access” is a very high-level term that can refer to a number of different sorts of attacks. The goal of these attacks is to access some resource that your machine should not provide the attacker. For example, a host might be a web server, and should provide anyone with requested web pages. However, that host should not provide command shell access without being sure that the person making such a request is someone who should get it, such as a local administrator.

4.6.2.1 Executing Commands Illicitly

It's obviously undesirable for an unknown and un-trusted person to be able to execute commands on your server machines. There are two main classifications of the severity of this problem: normal user access, and administrator access. A normal user can do a number of things on a system (such as read files, mail them to other people, etc.) that an attacker should not be able to do. This might, then, be all the access that an attacker needs. On the other hand, an attacker might wish to make configuration changes to a host (perhaps changing its IP address, putting a start-up script in place to cause the machine to shut down every time it's started or something similar). In this case, the attacker will need to gain administrator privileges on the host.

4.6.2.2 Confidentiality Breaches

We need to examine the threat model: what is it that you're trying to protect yourself against? There is certain information that could be quite damaging if it fell into the hands of a competitor, an enemy, or the public. In these cases, it's possible that compromise of a normal user's account on the machine can be enough to cause damage (perhaps in the form of PR, or obtaining information that can be used against the company, etc.)

While many of the perpetrators of these sorts of break-ins are merely thrill-seekers interested in nothing more than to see a shell prompt for your computer on their screen, there are those who are more malicious, as we'll consider next. (Additionally, keep in mind that it's possible that someone who is normally interested in nothing more than the thrill could be persuaded to do more: perhaps an unscrupulous competitor is willing to hire such a person to hurt you.)

4.6.2.3 Destructive Behavior

Among the destructive sorts of break-ins and attacks, there are two major categories.

Data Diddling--The data diddler is likely the worst sort, since the fact of a break-in might not be immediately obvious. Perhaps he's toying with the numbers in your spreadsheets, or changing the dates in your projections and plans. Maybe he's changing the account numbers for the auto-deposit of certain paychecks. In any case, rare is the case when you'll come in to work one day, and simply know that something is wrong. An accounting procedure might turn up a discrepancy in the books three or four months after the fact. Trying to track the problem down will certainly be difficult, and once that problem is discovered, how can any of your numbers from that time period be trusted? How far back do you have to go before you think that your data is safe?

Data Destruction--Some of those perpetrate attacks are simply twisted jerks who like to delete things. In these cases, the impact on your computing capability -- and consequently your business -- can be nothing less than if a fire or other disaster caused your computing equipment to be completely destroyed.

4.6.3 Where Do They Come From?

How, though, does an attacker gain access to your equipment? Through any connection that you have to the outside world. This includes Internet connections, dial-up modems, and even physical access. (How do you know that one of the temps that you've brought in to help with the data entry isn't really a system cracker looking for passwords, data phone numbers, vulnerabilities and anything else that can get him access to your equipment?)

In order to be able to adequately address security, all possible avenues of entry must be identified and evaluated. The security of that entry point must be consistent with your stated policy on acceptable risk levels.

4.6.4 Lessons Learned

From looking at the sorts of attacks that are common, we can divine a relatively short list of high-level practices that can help prevent security disasters, and to help control the damage in the event that preventative measures were unsuccessful in warding off an attack.

4.6.4.1 Hope you have backups

This isn't just a good idea from a security point of view. Operational requirements should dictate the backup policy, and this should be closely coordinated with a disaster recovery plan, such that if an airplane crashes into your building one night, you'll be able to carry on your business from another location. Similarly, these can be useful in recovering your data in the event of an electronic disaster: a hardware failure or a breakin that changes or otherwise damages your data.

4.6.4.2 Don't Put Data where it doesn't need to be

Although this should go without saying, this doesn't occur to lots of folks. As a result, information that doesn't need to be accessible from the outside world sometimes is, and this can needlessly increase the severity of a break-in dramatically.

4.6.4.3 Avoid Systems with Single Points of Failure

Any security system that can be broken by breaking through any one component isn't really very strong. In security, a degree of redundancy is good, and can help you protect your organization from a minor security breach becoming a catastrophe.

4.6.4.4 Stay Current with Relevant Operating System Patches

Be sure that someone who knows what you've got is watching the vendors' security advisories. Exploiting old bugs is still one of the most common (and most effective!) means of breaking into systems.

4.6.4.5 Watch for Relevant Security Advisories

In addition to watching what the vendors are saying, keep a close watch on groups like CERT and CIAC. Make sure that at least one person (preferably more) is subscribed to these mailing lists

4.6.4.6 Have Someone on Staff be Familiar with Security

Having at least one person who is charged with keeping abreast of security developments is a good idea. This need not be a technical wizard, but could be someone who is simply able to read advisories issued by various incident response teams, and keep track of various problems that arise. Such a person would then be a wise one to consult with on security related issues, as he'll be the one who knows if web server software version such-and-such has any known problems, etc.

4.7 Generation and Distribution of Keys

A constant supply of key bases is required to keep the distributed network system operating. One possible plan is shown in Figure 4.7, in which two major key preparation stations are depicted, one in A and other is B. Each such station contains a large general-purpose computer with about six tape units. Each key preparation site uses separately written, highly complex, random number generating programs. Three individuals at each site working independently insert choice parameters, which modify the random number generator. Conventional one-inch magnetic computer tapes, recorded at high speed, are played back into a 1/4" tape duplicator for preparation of the 300-ft spools of 1/4" tape used in the Multiplexing Stations. The one-inch computer tape outputs are also used to drive an off-line cardpunch to prepare the shorter set of keys, which are used by the Switching Nodes. The output of each of the two sites' tapes and card duplicating facilities are stored in about twenty geographically distributed sites.

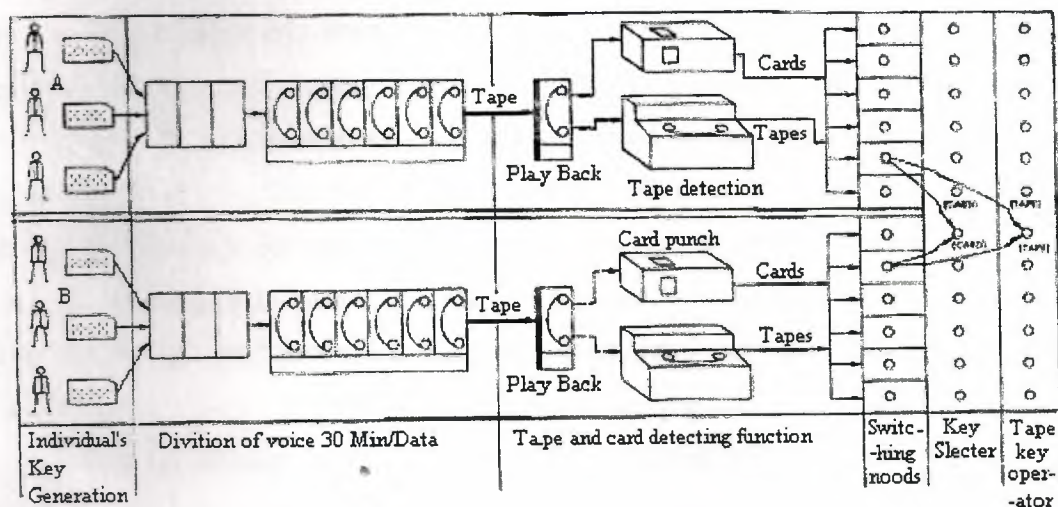


Figure 4.7: One method of distributing the keys

The Switching Node and Multiplexing Station keys are comprised of two parts, one coming from the distribution site prepared by the A unit, and the other part coming from the B unit via different distribution sites. Each member of a two-man team has mechanical key access to only his own part of the key base. Thus, the system is relatively secure from a single enemy agent having access to an entire key base for any unit.

4.8 Modification of Derived Key Base

To this point, both Multiplexing Stations are synchronized and are using the same derived key bases. In it the same derived key base is used for more than a single conversation call. After the setup interval, Message Blocks will arrive at a very high rate. It is necessary to create a key from the derived key base at a very rapid rate, leaving very little time for processing. As this is a routine continuous operation, a "stamping mill" processor, with a portion of the Multiplexing Station equipment working full time on this operation, is utilized. The Multiplexing Station uses a drum or similar recalculating register to store the key bases, derived keys, and the Message Blocks. The processing scheme used depends primarily upon a very low Message Block error rate at the Multiplexing Stations. Unfiltered errors and lost Message Blocks are expected to be such rare events, that we shall intentionally "knock down" a quasi-circuit if a single bad Message Block slips by the error-detection filters. Incoming encrypted message alternately fills one of the two assigned registers while the other register is simultaneously being read out and "logically-added" to the key base. The clear output message is then stored on one of two alternately assigned registers reserved for this purpose. Meanwhile, the clear message and the incoming message operate upon one another in a controlled manner to produce a new key base, based upon the previous key base used. This procedure may appear to be similar to the conventional "auto key" procedure, but it should be noted that the next key is related to its previous one by a very complex and unknown mathematical operation. Even having the entire encrypted message and a sample of clear message will not facilitate ascertaining subsequent samples of clear message. Thus, very high speed processing of Message Blocks with high cryptographic security for 1024 separate subscribers per Multiplexing Station does not appear particularly difficult to accomplish.

It should be pointed out that the detailed implementation described may or may not be the precise method used. The present detailed description seeks only to point out that secure cryptographic processing at extremely high data rates appears technically possible.

CONCLUSION

In computer security, public key is one of several ways of achieving isolation, and is not one of the more important ways. In network security, cryptography is the main attraction - everything is done via message-passing (ultimately), so the only secure way to achieve confidentiality and the authentication needed for access control is through cryptography. Security is a very difficult topic.

Computer security threats were rare, and were basically concerned with insiders; these threats were well understood and dealt with using standard techniques. The key for building a secure network is to define what security means to your organization.

Everyone has a different idea of what "security" as there are many ways of encrypt and decrypt data so no body can decide the acceptable levels of risk. The key for building a secure network is depend on the public key functions and size of key chosen. Once that has been defined, everything that goes on with the network can be evaluated with respect to that policy. Many people pay great amounts of lip service to security, but do not want to be bothered with it when it gets in their way.

REFERENCES

- [1] Buzzard, James. The Client-Server Paradigm: Making Sense Out of the Claims, Data Based Advisor, August 1990.
- [2] McGoveran, David. Evaluating an RDBMS for Client/Server Applications. Alternative Technologies, 1992.
- [3] Adler, R. M. "Distributed Coordination Models for Client/Sever Computing." Computer 28, April 1995, pp. 14-22.
- [4] Architecture. A Comparison of Two-Tier and Three-Tier Systems." *Information Systems Management Magazine*", Spring 1996.
- [5] "Remapping the Computing Environment for the 90s: The Impact of Client/Server," Sybase Executive Summary, Winter 1993.
- [6] Lloyd Taylor, "comp. Client-ServerFAQ maintainer", "<http://www.wp.com/Lloyd.Taylor>". 1999.
- [7] Schussel, George. "*Client/Server Past, Present, and Future*", "<http://www.dciexpo.com/geos>", 1995.
- [8] Queensland University of Technology, "Oscar - DSTC's Public key Infrastructure . Retrieved December 5, 2003.<http://oscar.dstc.qut.edu.au/Project>," December 1998,
- [9] Data Encryption Standard, Federal Information Processing Standard (FIPS) Publication 46, National Bureau of Standards, U.S. Department of Commerce, Washington D.C. (January 1977).
- [10] 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.