NEAR EAST UNIVERSITY

Faculty of Engineering

Department of Computer Engineering

CRYPTOGRAPHY & SECURITY OVER NETWORK

Graduation Project
Com 400

Prepared By: Mohammad Maslat (20010690).

Supervisor: Assoc. Prof. Dr. Rahib Abyev.

Nicosia - 2006

Dedicated to my parents

LIST OF ABBREVIATIONS

MDC:	Modification Detection Codes.
MAC:	Message Authentication Codes.
ECB:	Electronic Codebook.
CBC:	Cipher Block Chaining.
DES:	Data Encryption Standard.
LFSR:	Linear Feedback Shift Register.
SPEKE:	Simple Password Exponential key Exchange.
DH-EKE:	Diffie-Hellman Encrypted Key Exchange.
DSA:	Digital Signature Methods.
FFT:	Fast Fourier Transform.
PGP:	Pretty Good Privacy.
RSA:	Rivest, Shamir, Adleman.
KE:	Keyless Encryption.
OTP:	One-Time Pads.
RSAP:	RSA Problem.
ISO:	International Standards Organization.
OSI:	Open Systems Interconnect.
TCP/IP:	Transport Control Protocol/Internet Protocol.
UDP:	User Datagram Protocol.
IETF:	Internet Engineering Task Force.

LIST OF ABBREVIATIONS

DNS:	Domain Name System.
ISP:	Internet Service Provider.
HTTP:	Hypertext Transfers Protocol.
SMTP:	Simple Mails Transfer Protocol.
DoS:	Denial-of-Service.
DMZ:	Demilitarized Zone.
FTP:	File Transfer Protocol.
ACL:	Access Control lists.
NAT:	Network Address Translation.
PAT:	Port Address Translation.
VPN:	Virtual Private Networks.

ACKNOWLEDGEMENT

First of all I would like to thanks Allah {God} for guiding me through my studies And who has given me the power and the patience to finish my bachelor degree's studies successfully.

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

Also, I feel proud to pay my regards to my project adviser "Assoc. Prof. Dr. Rahib Abyev ". He never disappointed me in any affair. He delivered me too much information and did his best of efforts to make me able to complete my project. Not to forget to give my thanks to the ***NEAR EAST UNIVERSITY*** *education staff especially to the computer engineering doctors for their helping to take this degree and to achieve this level of education.*

I will never forget the days that I have been in Cyprus, from the University to the good friends that I have enjoyed my 4 years with them. I would like to thank them for there kindness in helping me to complete my project:

My close friends; Eng.Baha Khalaf, Eng. Ala Mansour, Eng.Sa'ed Maslat and Also: Eng.Talal Khader, Khaled Abu Zagleh (Abu Sharks), AbdulLAtif Al Shamali, Omar Enbawi, Qusi, Mazen al Shawabkeh, Fadi Hamad, Tamer Maiah.

Also my special thanks to my all friends in Jordan.

ACKNOWLEDGEMENT

At the end I would like to thank my family again starting with my great Father: {ASA'D MASLAT}, the best mother in the world: {IBTISAM MASLAT}, As well as thanks to my brother Moayyad Maslat For his encourage, my sisters and not to forget their husbands, and best relatives (MOHAMMAD AL SHADFAN & NEDAL AL RABAY'A), I would like to give them my regards and appreciated their excellency and efforts with asking about me.

Mohammad Maslat.

3/2/2006

ABSTRACT

Cryptography algorithms are applied to protect a message or file from being read by network hackers, eavesdroppers. The encryption programs encrypt the text and will change the letters into symbols and other weird characters, so when someone opens the file they cannot read it. The interconnection of networks is an increasing trend in government and private industry. There is the obvious danger that connections made in such an extended network may increase the risk of a security compromise, with the owners unaware of the risk.

Network connections should therefore be protected, at a level based on the risk. The assumption must be that the connecting parties are to a certain degree hostile and have to be strictly constrained to the access for which the connection was agreed.

Although cryptography is fascinating and glamorous, because of its association with such things as espionage, diplomacy, and the higher levels of the military, it has a limited but important role in the area of network security.

The RSA cryptosystem, named after its inventors R. Rivest, A. Shamir, and L. Adleman, is the most widely used public-key cryptosystem. It may be used to provide both secrecy and digital signatures and its security is based on the intractability of the integer factorization. RSA encryption is most commonly used for the transport of symmetric-key encryption algorithm keys and for the encryption of small data items. The RSA cryptosystem has been patented in the U.S. and Canada. Several standards organizations have written, or are in the process of writing, standards that address the use of the RSA cryptosystem for encryption, digital signatures, and key establishment. For discussion of patent and standards issues related to RSA. The description of RSA algorithm is given in the thesis.

TABLE OF CONTENTS

Introduction

The origin of the word cryptology lies in ancient Greek. The word cryptology is made up of two components: "kryptos", which means hidden and "logos" which means word. Cryptology is as old as writing itself, and has been used for thousands of years to safeguard military and diplomatic communications. For example, the famous Roman emperor Julius Caesar used a cipher to protect the messages to his troops. Within the field of cryptology one can see two separate divisions: cryptography and cryptanalysis. The cryptographer seeks methods to ensure the safety and security of conversations while the cryptanalyst tries to undo the farmer's work by breaking his systems.

The main goals of modern cryptography can be seen as: user authentication, data authentication (data integrity and data origin authentication), non-repudiation of origin, and data confidentiality. In the following section we will elaborate more on these services. Subsequently we will explain how these services can be realized using cryptographic primitives.

A cryptographic system (or a cipher system) is a method of hiding data so that only certain people can view it. Cryptography is the practice of creating and using cryptographic systems. Cryptanalysis is the science of analyzing and reverse engineering cryptographic systems. The original data is called plaintext. The protected data is called cipher text. Encryption is a procedure to convert plaintext into cipher text. Decryption is a procedure to convert cipher text into plaintext. A cryptographic system typically consists of algorithms, keys, and key management facilities. There are two basic types of cryptographic systems: symmetric ("private key") and asymmetric ("public key").

Symmetric key systems require both the sender and the recipient to have the same key. This key is used by the sender to encrypt the data, and again by the recipient to decrypt the data. Key exchange is clearly a problem. How do you securely send a key that will enable you to send other data securely? If a private key is intercepted or stolen, the adversary can act

as either party and view all data and communications. You can think of the symmetric crypto system as akin to the Chubb type of door locks. You must be in possession of a key to both open and lock the door. Asymmetric cryptographic systems are considered much more flexible. Each user has both a public key and a private key.

Messages are encrypted with one key and can be decrypted only by the other key. The public key can be published widely while the private key is kept secret. If Alice wishes to send Bob a secret, she finds and verifies Bob's public key, encrypts her message with it, and mails it off to Bob. When Bob gets the message, he uses his private key to decrypt it. Verification of public keys is an important step. Failure to verify that the public key really does belong to Bob leaves open the possibility that Alice is using a key whose associated private key is in the hands of an enemy. Public Key Infrastructures or PKI's deal with this problem by providing certification authorities that sign keys by a supposedly trusted party and make them available for download or verification. Asymmetric ciphers are much slower than their symmetric counterparts and key sizes are generally much larger. You can think of a public key system as akin to a Yale type door lock. Anyone can push the door locked, but you must be in possession of the correct key to open the door.

The project is devoted the description of cryptographic algorithms, particularly RSA algorithm over network. The Goal of RSA Algorithm is to implement a demonstrable application that will perform the encryption and decryption of a text file using RSA Algorithm. I will give input as plaintext and it will generate the corresponding ciphertext. Ciphertext is decrypted to get the original plain text.

R.S.A. stands for Rivest, Shamir and Adleman - the three cryptographers who invented the first practical commercial public key cryptosystem. Today it is used in web browsers, email programs, mobile phones, virtual private networks, secure shells, and many other places. Exactly how much security it provides is debatable, but with sufficiently large keys you can be confident of foiling the vast majority of attackers. Until recently the use of RSA was very much restricted by patent and export laws. However, the patent has now expired and US export laws have been relaxed.

CHAPTER ONE

1. OVERVIEW OF CRYPTOGRAPHY SYSTEMS

1.1 Introduction

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 mentioned.

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 fsecurity 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 What Does Cryptography mean

Cryptography means 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.

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.

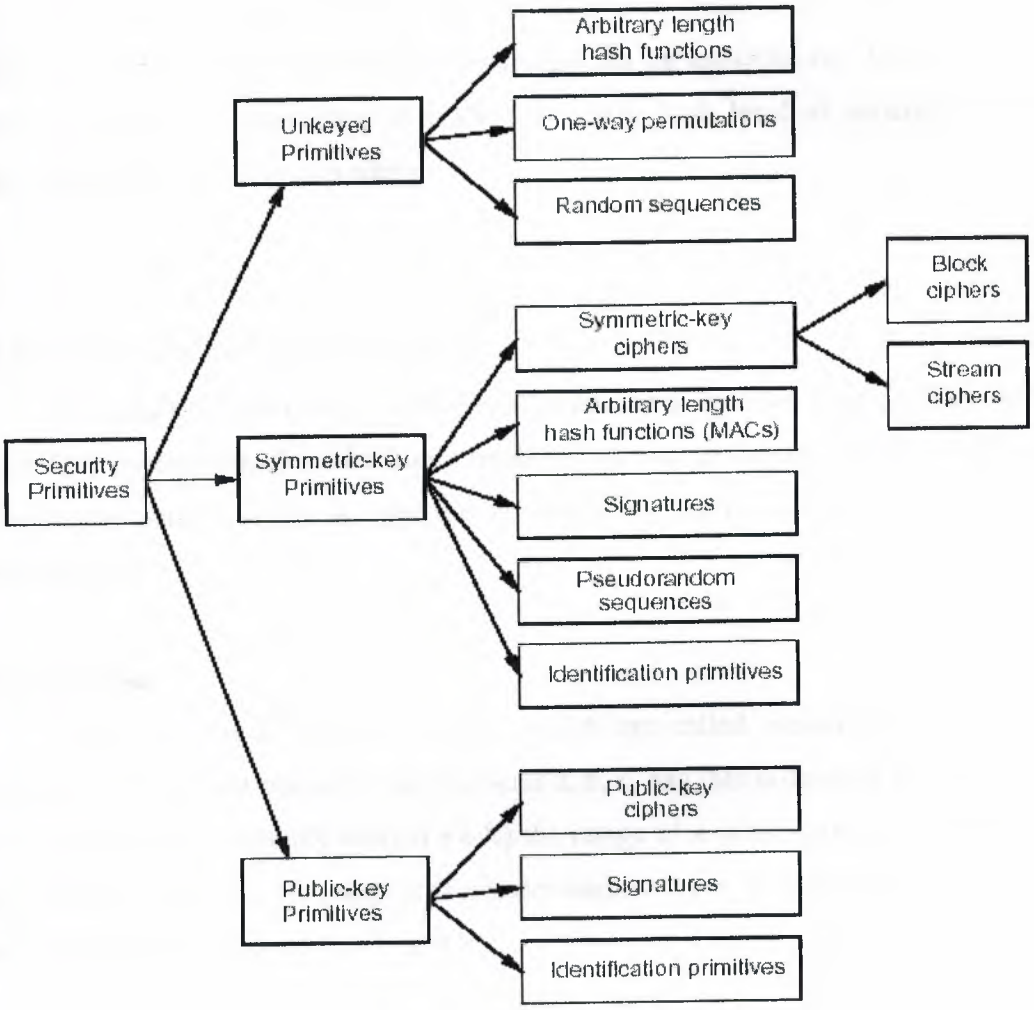

Figure 1.1 A taxonomy of cryptographic primitives.

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→Y.

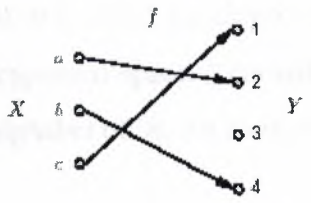

Figure 1.2 A function f from a set X to a set Y.

- 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→Y is 1-1 and Im (f) = Y, then f is called a bijection.

- One-way functions: A function f from a set X to a set Y is called a one-way function if f (x) is easy to compute for all $x \in X$ but for essentially all elements $y \in$ 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→S).
- Involutions: Involutions have the property that they are their own inverses. (i.e., f: S→ 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.
- C denotes a set called the cypertext space. C consists of strings of symbols from an alphabet; differ from the alphabet of $\mathcal{M}$. An element of C is called a cypertext.

1.3.2.2 Encryption and Decryption Transformations

- $\mathcal{K}$ 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 C, denoted by $\mathcal{E}e$.
- $\mathcal{D}_d$ denotes a bijection from 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 cypertext 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 ∈ M to Bob, she computes c = Ee (m) and transmits this to Bob. Upon receiving c, Bob computes D_d (c) = m and hence recovers the original message m.

The question arises as to why keys are necessary. If some particular encryption/decryption transformation is exposed then one does not have to redesign the entire scheme but simply change the key. Figure 1.3 provides a simple model of a two-party communication using encryption.

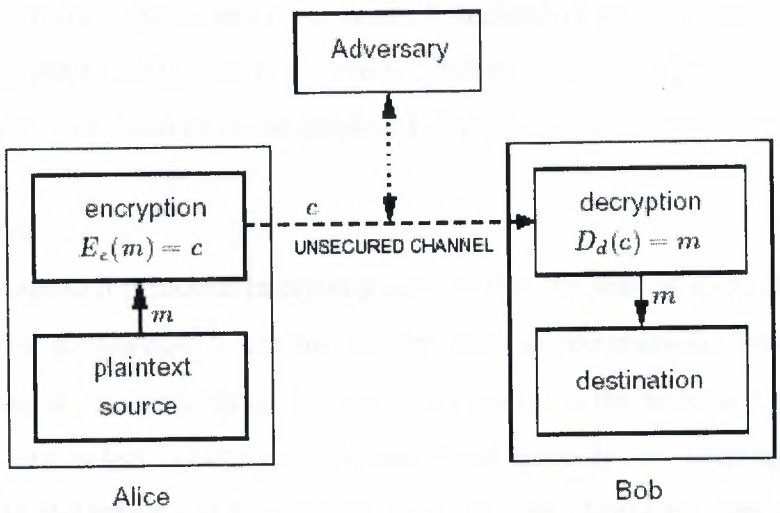

Figure 1.3 Schematic of a two-party communication.

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 $\mathcal{M}$; 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 cypertext within some appropriate time frame.

Trying all possible keys to see which one the communicating parties are using can break an encryption scheme. 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.

1.4 Symmetric-key Encryption

Consider an encryption scheme consisting of the sets of encryption and decryption transformations $\{\mathcal{E}e: e \in \mathcal{K}\}$ and $\{\mathcal{D}_d : d \in \mathcal{K}\}$, 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.

The block diagram of Figure 1.4, with the addition of the secure channel, can describe a two-party communication using symmetric-key encryption.

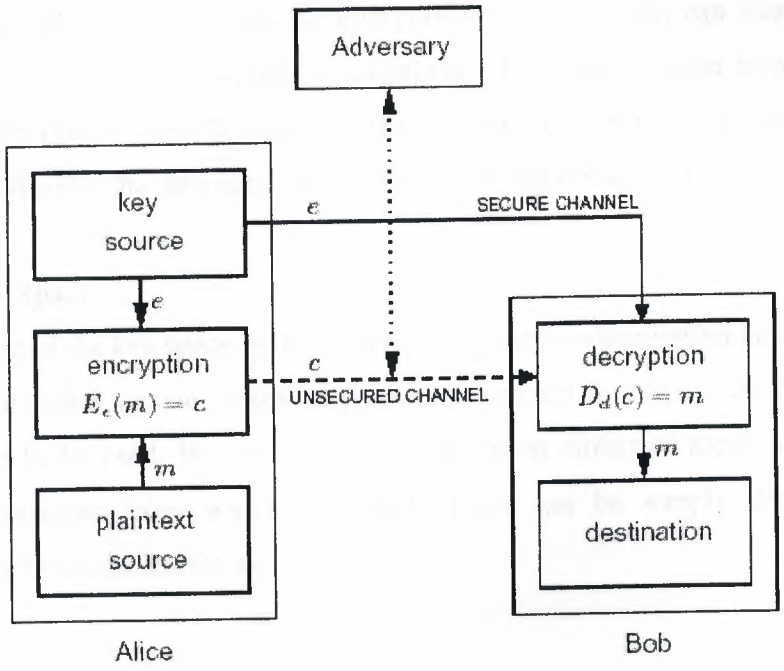

Figure 1.4 Two-party communication using encryption, with a secure channel

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 $S_{\mathcal{A}}$ and $V_{\mathcal{A}}$ provide a digital signature scheme for $\mathcal{A}$.

- $\mathcal{M}$ 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.
- $S_{\mathcal{A}}$ is a transformation from the message set $\mathcal{M}$ to the signature set $\mathcal{S}$, and is called a signing transformation for entity $\mathcal{A}$.

- $\mathcal{V}_A$ is a transformation from the set $\mathcal{M} \times S$ to the set {true, false} $\mathcal{V}_A$ 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 $\{\mathcal{E}e: e \in \mathcal{K}\}$ be a set of encryption transformations, and let $\{\mathcal{D}_d: d \in \mathcal{K}\}$ be the set of corresponding decryption transformations, where $\mathcal{K}$ is the key space. Consider any pair of associated encryption/decryption transformations $(\mathcal{E}e; \mathcal{D}d)$ and suppose that each pair has the property that knowing $\mathcal{E}e$ it is computationally infeasible, given a random ciphertext $c \in C$, to find the message $m \in \mathcal{M}$ such that $\mathcal{E}e(m) = c$. This property implies that given e it is infeasible to determine the corresponding decryption key d. $\mathcal{E}e$ 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

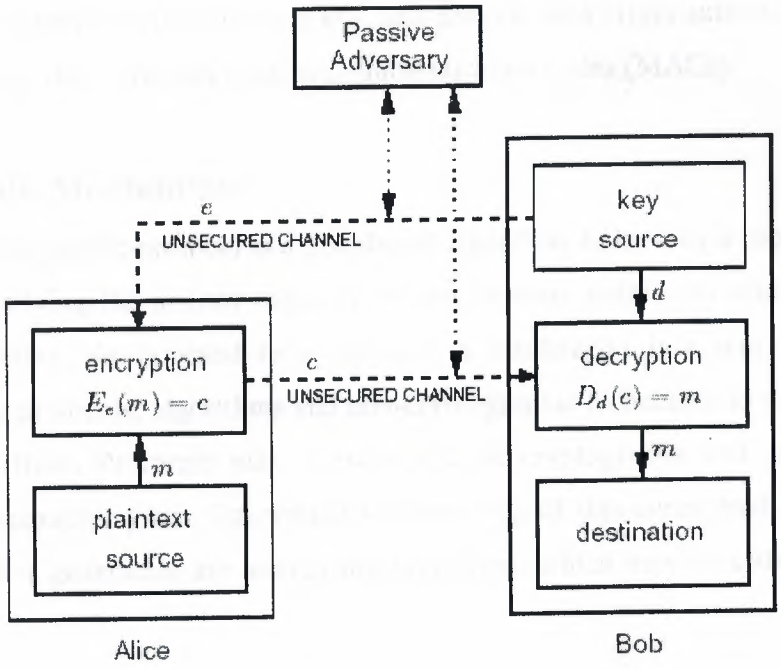

Figure 1.5 Encryption using public-key techniques.

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.

CHAPTER TWO

2. CRYPTOGRAPHY FUNCTIONS

2.1 Overview

In this chapter basic functions involved in cryptography are explained. Functions that 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.

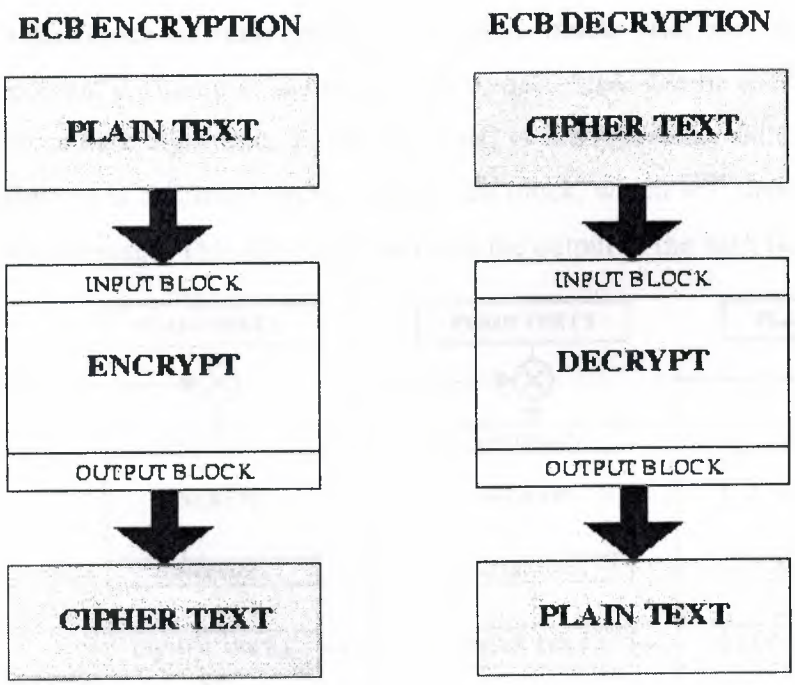

Figure 2.1: Shows a ECB Encryption/Decryption Model

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.

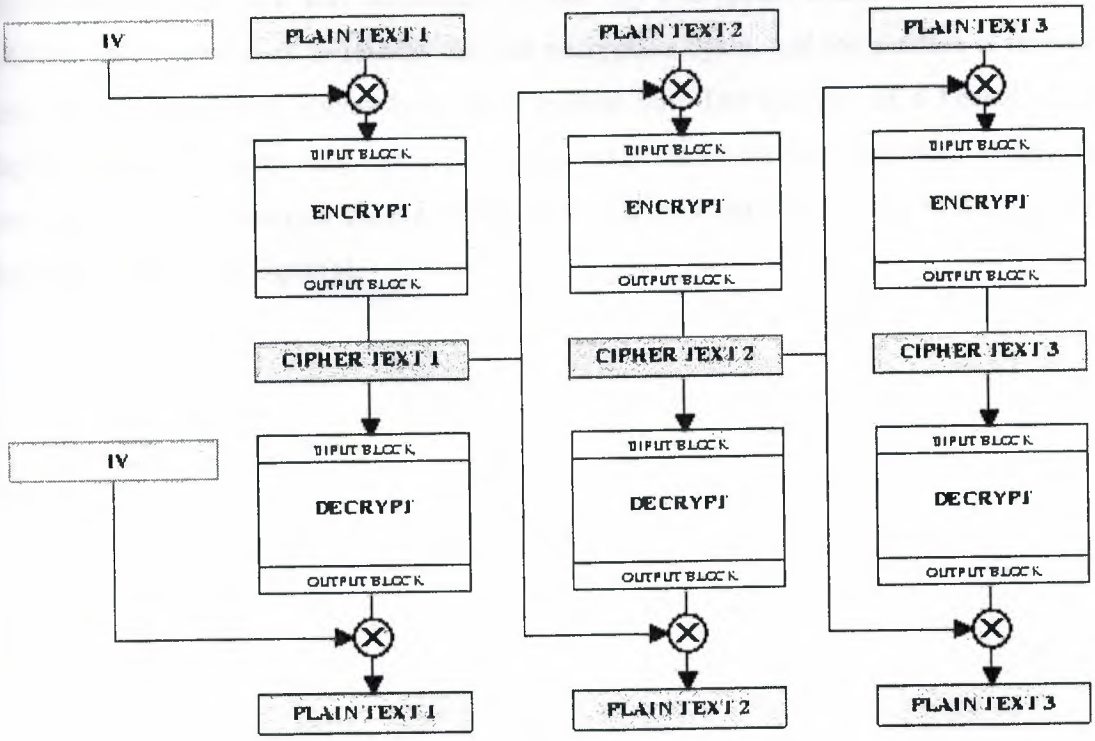

Figure 2.2: Shows a CBC Encryption/Decryption Model

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).

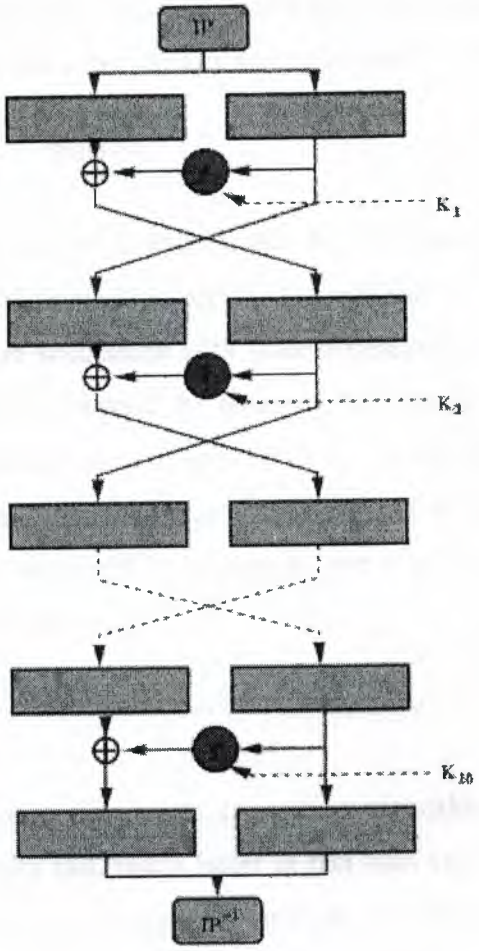

Figure 2.3: Shows a Feistel Model

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 combining the keystream with the plaintext, usually with the bitwise XOR operation, provides encryption. 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.

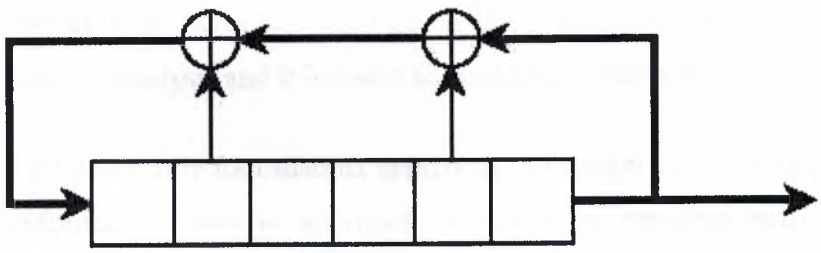

Figure 2.1: Shows a Linear Feed Back Register Model

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

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

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→B: h(h(K))
S6.	Bob verifies h(h(K)) is correct,	B→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:	$Q_A = g^{R}{}_A \bmod p$,	$A \rightarrow B$: $E_S (Q_A)$.
D2.	Bob computes:	$Q_B = g^{R}{}_B \bmod p$,	$B \rightarrow A$: $E_S (Q_B)$.
D3.	Alice computes:	$K = h(Q_B{}^{R}{}_A \bmod p)$	
D4.	Bob computes:	$K = h(Q_A{}^{R}{}_B \bmod p)$	

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 for 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 ($g^R{}_X$ mod 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:

$$K' = (E_{Si}^{-1}(E_S(g^{R}{}_B)))^{RA}$$

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:

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

$$= 2^{(S R_A)(X/S')} \bmod p$$

$$= 2^{(X R_A S/S')} \bmod p$$

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

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

Thus, when S' = S, K' = K. More generally, the attack works because the dictionary of passwords $\{S_1, S_2 \ldots S_n\}$ is equivalent to a dictionary of exponents $E = \{e_1, e_2 \ldots e_n\}$, 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 gc 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.

CHAPTER THREE

3. ENCRYPTION & DECRYPTION USING RSA ALGORITHM

3.1 Overview

RSA is a public-key cryptosystem developed by MIT professors: Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman in 1977 in an effort to help ensure Internet security. As Steve Burnett of RSA Data Security, Inc. described it, a cryptosystem is simply an algorithm that can convert input data into something unrecognizable (encryption), and convert the unrecognizable data back to its original form (decryption).

The RSA scheme is a block cipher in which the plaintext and cipher text are integers between 0 and n-1 for some n. A typical size for n is 1024 bits, or 309 decimal digits.

To encrypt data, enter the data ("plaintext") and an encryption key to the encryption portion of the algorithm. To decrypt the "cipher text," a proper decryption key is used at the decryption portion of the algorithm. Those keys, which contain simply a string of numbers, are called public key and private key, respectively. For example, suppose Alice intends to send e-mail to Bob. Through a public-key directory, she finds his public key. Then, she encrypts her message using the key and sends it to Bob. This public key, however, will not decrypt the cipher text. Knowledge of Bob's public key will not help an eavesdropper. In order for Bob to decrypt his cipher text, he must use his private key. If Bob wants to respond to Alice, he encrypts his message using her public key.

The challenge of public-key cryptography is developing a system in which it is impossible to determine the private key. This is accomplished through the use of a one-way function. With a one-way function, it is relatively easy to compute a result given some input values. In mathematical terms, given x, computing f(x) is easy, but given f(x), computing x is nearly impossible. The one-way function used in RSA is multiplication of prime numbers. It is easy to multiply two big prime numbers, but for most very large

primes, it is extremely time-consuming to factor them. Public-key cryptography uses this function by building a cryptosystem that uses two large primes to build the private key and the product of those primes to build the public key.

3.2 How does cryptographic algorithm work?

A *cryptographic algorithm,* or cipher, is a mathematical function used in the encryption and decryption process. A cryptographic algorithm works in combination with a *key* — a word, number, or phrase to encrypt the plaintext. The same plaintext encrypts to different cipher text with different keys. The security of encrypted data is entirely dependent on two things: the strength of the cryptographic algorithm and the secrecy of the key. A cryptographic algorithm, plus all possible keys and all the protocols that make it work comprise a *cryptosystem.* PGP is a cryptosystem.

3.3 Different types of Cryptosystems/Encryptions:

There are four ways of encryption that we mostly use in network system

- Pretty Good Privacy, PGP
- Rivest, Shamir, Adleman, RSA
- Keyless Encryption, KE
- One-Time Pads, OTP

3.3.1 PGP (Pretty Good Privacy)

Pretty Good Privacy. Up to this point we've talked about private-key cryptography (one key used y both parties). There was one problem with this kind of encryption: If the key was intercepted, a third party could decrypt the messages. So, the ideas of public-key cryptography were developed. Here's how it works...

Everyone has two keys: a public and a private key. When someone wants to send something to a recipient, they (the sender) encrypt it with the recipient's public key. Then the only way to decrypt it is with the recipient's private key. One of the other benefits to PGP is that it allows the sender to "sign" their messages. This proves that the message came from the sender and has not been altered in transport. Based on this theory, PGP

allows everyone to publicize their public keys, while keeping their private keys secret. The result is that anyone can encrypt a message to someone else, as long as they have that person's public key.

In actuality, PGP uses a series of private key, public key and one-way hash functions to encrypt a message. A one-way hash function takes some plaintext and translates it into a specific hash. The hash is unique to the message (like a fingerprint is to a person). The hash is also non-reversible, hence the name one-way. Let's run through an example of what PGP does to encrypt and decrypt an e-mail message. Our sender will be Chris and our receiver will be Brian.

- Chris writes his message.
- Chris uses a one-way hash function (such as MD5) to create a hash for the message.
- Chris, via RSA or some other digital signature algorithm, signs the hash with his private key.
- Chris merges the message and the signature, resulting in a new-signed message.
- A random encryption key is generated, the session key.
- Chris uses the session key to encrypt the message, using DES or some other private key method.
- Chris gets Brian's public key.
- Chris then encrypts the key with Brian's public key, via RSA or some other public key method.
- Chris merges the encrypted message and the encrypted key and mails it to Brian.

Once Brian receives the message he can have PGP decrypt it. Here's what it would do:

- Brian separates the encrypted message and the encrypted session key.
- Using RSA, Brian decrypts the session key.
- Using DES, Brian decrypts the message with the decrypted session key.
- Brian then separates the message and the signature.

- Using MD5, Brian calculates the hash value of the message.
- Brian gets Chris' public key.
- Via RSA, and Chris' public key, Brian decrypts the signature.
- Brian then compares the hash value and the decrypted signature. If they are the same, Brian knows that the message is authentic and has not been altered since Chris signed it.
-

3.3.2 RSA (Rivest, Shamir, Adleman)

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

$$C = M^e \bmod n$$

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

$$M = C^d \bmod n$$

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

values through equation (2) and get 19, our original plaintext! Try it yourself with other numbers.

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

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

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

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

3.4 The RSA Algorithm

Let's run the RSA algorithm through steps of how RSA algorithm procedure does work:

3.4.1 Key Generation

1.Generate two large prime numbers, p and q

2. Let n = pq

3. Let m = (p-1)(q-1)

4. Choose a small number e, coprime to m

5. Find d, such that de % m = 1

Publish e and n as the public key, Keep d and n as the secret k

Encryption

$C = P^e \% n$

Decryption

$P = C^d \% n$

x % y means the remainder of x divided by y

The reasons why this algorithm works are discussed in the mathematics section. Its security comes from the computational difficulty of factoring large numbers. To be secure, very large numbers must be used for p and q - 100 decimal digits at the very least.

I'll now go through a simple worked **example 1.**

a) Key Generation

1) Generate two large prime numbers, p and q

To make the example easy to follow I am going to use small numbers, but this is not secure. To find random primes, we start at a random number and go up ascending odd numbers until we find a prime. Lets have:

p = 7
q = 19

2) Let n = pq
n = 7 * 19
= 133

3) Let m = (p - 1)(q - 1)
m = (7 - 1)(19 - 1)
= 6 * 18
= 108

4) Choose a small number, e coprime to m

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

e = 2 => gcd(e, 108) = 2 (no)

e = 3 => gcd(e, 108) = 3 (no)

e = 4 => gcd(e, 108) = 4 (no)

e = 5 => gcd(e, 108) = 1 (yes!)

5) Find d, such that de % m = 1

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

n = 0 => d = 1 / 5 (no)

n = 1 => d = 109 / 5 (no)

n = 2 => d = 217 / 5 (no)

n = 3 => d = 325 / 5

= 65 (yes!)

To do this with big numbers, a more sophisticated algorithm called extended Euclid must be used.

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

b) Encryption

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

$C = P^e \% n$
$= 6^5 \% 133$
$= 7776 \% 133$
$= 62$

c) Decryption

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

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

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

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

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

3.5 From Applied Cryptography

$n = pq$

$ed \_ 1 \bmod (p \_ 1)(q \_ 1)$

$c = me \bmod n$

$m = cd \bmod n$

n is the product of p and q, two prime numbers.

The product of e and d divided by (p_1)(q _1) has a remainder of 1. C is the remainder of me divided by n; m is the remainder of cd divided by n.

3.5.1 The Product of Two Primes

n is the product of two prime numbers, p and q. n is made public as it is used in decryption as well as encryption. p and q should never be revealed.

Example:

p = 59, q = 67; n = 59 _ 67 = 3953

3.5.2 e and d, The Keys

e, the encryption key, is a random number relatively prime to (p _ 1)(q _ 1). d can be calculated to be de = 1 + x(p _ 1)(q _ 1), x is a throwaway value as it is \lost" in the modulus calculation. e is the public key and d is the private key.

Example, continued:

e = 7, (p _ 1)(q _ 1) = 3828

de = 1 + x(p _ 1)(q _ 1)

xd = 547, x = 1, d = 547

e = 7, d = 547

3.6 Key types

Two key types are employed in the primitives and schemes defined: *RSA public key* and *RSA private key*. Together, an RSA public key and an RSA private key form an *RSA key pair*.

This specification supports so-called "multi-prime" RSA where the modulus may have more than two prime factors. Better performance can be achieved on single processor platforms, but to a greater extent on multiprocessor platforms, where the modular exponentiations involved can be done in parallel.

NEAR EAST UNIVERSITY

Faculty of Engineering

Department of Computer Engineering

CRYPTOGRAPHY & SECURITY OVER NETWORK

Graduation Project
Com 400

Prepared By: Mohammad Maslat (20010690).

Supervisor: Assoc. Prof. Dr. Rahib Abyev.

Nicosia - 2006

Dedicated to my parents

LIST OF ABBREVIATIONS

MDC:	Modification Detection Codes.
MAC:	Message Authentication Codes.
ECB:	Electronic Codebook.
CBC:	Cipher Block Chaining.
DES:	Data Encryption Standard.
LFSR:	Linear Feedback Shift Register.
SPEKE:	Simple Password Exponential key Exchange.
DH-EKE:	Diffie-Hellman Encrypted Key Exchange.
DSA:	Digital Signature Methods.
FFT:	Fast Fourier Transform.
PGP:	Pretty Good Privacy.
RSA:	Rivest, Shamir, Adleman.
KE:	Keyless Encryption.
OTP:	One-Time Pads.
RSAP:	RSA Problem.
ISO:	International Standards Organization.
OSI:	Open Systems Interconnect.
TCP/IP:	Transport Control Protocol/Internet Protocol.
UDP:	User Datagram Protocol.
IETF:	Internet Engineering Task Force.

LIST OF ABBREVIATIONS

DNS:	Domain Name System.
ISP:	Internet Service Provider.
HTTP:	Hypertext Transfers Protocol.
SMTP:	Simple Mails Transfer Protocol.
DoS:	Denial-of-Service.
DMZ:	Demilitarized Zone.
FTP:	File Transfer Protocol.
ACL:	Access Control lists.
NAT:	Network Address Translation.
PAT:	Port Address Translation.
VPN:	Virtual Private Networks.

ACKNOWLEDGEMENT

First of all I would like to thanks Allah {God} for guiding me through my studies And who has given me the power and the patience to finish my bachelor degree's studies successfully.

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

Also, I feel proud to pay my regards to my project adviser "Assoc. Prof. Dr. Rahib Abyev ". He never disappointed me in any affair. He delivered me too much information and did his best of efforts to make me able to complete my project. Not to forget to give my thanks to the ***NEAR EAST UNIVERSITY*** *education staff especially to the computer engineering doctors for their helping to take this degree and to achieve this level of education.*

I will never forget the days that I have been in Cyprus, from the University to the good friends that I have enjoyed my 4 years with them. I would like to thank them for there kindness in helping me to complete my project:

My close friends; Eng.Baha Khalaf, Eng. Ala Mansour, Eng.Sa'ed Maslat and Also: Eng.Talal Khader, Khaled Abu Zagleh (Abu Sharks), AbdulLAtif Al Shamali, Omar Enbawi, Qusi, Mazen al Shawabkeh, Fadi Hamad, Tamer Maiah.

Also my special thanks to my all friends in Jordan.

ACKNOWLEDGEMENT

At the end I would like to thank my family again starting with my great Father: {ASA'D MASLAT}, the best mother in the world: {IBTISAM MASLAT}, As well as thanks to my brother Moayyad Maslat For his encourage, my sisters and not to forget their husbands, and best relatives (MOHAMMAD AL SHADFAN & NEDAL AL RABAY'A), I would like to give them my regards and appreciated their excellency and efforts with asking about me.

Mohammad Maslat.

3/2/2006

ABSTRACT

Cryptography algorithms are applied to protect a message or file from being read by network hackers, eavesdroppers. The encryption programs encrypt the text and will change the letters into symbols and other weird characters, so when someone opens the file they cannot read it. The interconnection of networks is an increasing trend in government and private industry. There is the obvious danger that connections made in such an extended network may increase the risk of a security compromise, with the owners unaware of the risk.

Network connections should therefore be protected, at a level based on the risk. The assumption must be that the connecting parties are to a certain degree hostile and have to be strictly constrained to the access for which the connection was agreed.

Although cryptography is fascinating and glamorous, because of its association with such things as espionage, diplomacy, and the higher levels of the military, it has a limited but important role in the area of network security.

The RSA cryptosystem, named after its inventors R. Rivest, A. Shamir, and L. Adleman, is the most widely used public-key cryptosystem. It may be used to provide both secrecy and digital signatures and its security is based on the intractability of the integer factorization. RSA encryption is most commonly used for the transport of symmetric-key encryption algorithm keys and for the encryption of small data items. The RSA cryptosystem has been patented in the U.S. and Canada. Several standards organizations have written, or are in the process of writing, standards that address the use of the RSA cryptosystem for encryption, digital signatures, and key establishment. For discussion of patent and standards issues related to RSA. The description of RSA algorithm is given in the thesis.

TABLE OF CONTENTS

Introduction

The origin of the word cryptology lies in ancient Greek. The word cryptology is made up of two components: "kryptos", which means hidden and "logos" which means word. Cryptology is as old as writing itself, and has been used for thousands of years to safeguard military and diplomatic communications. For example, the famous Roman emperor Julius Caesar used a cipher to protect the messages to his troops. Within the field of cryptology one can see two separate divisions: cryptography and cryptanalysis. The cryptographer seeks methods to ensure the safety and security of conversations while the cryptanalyst tries to undo the farmer's work by breaking his systems.

The main goals of modern cryptography can be seen as: user authentication, data authentication (data integrity and data origin authentication), non-repudiation of origin, and data confidentiality. In the following section we will elaborate more on these services. Subsequently we will explain how these services can be realized using cryptographic primitives.

A cryptographic system (or a cipher system) is a method of hiding data so that only certain people can view it. Cryptography is the practice of creating and using cryptographic systems. Cryptanalysis is the science of analyzing and reverse engineering cryptographic systems. The original data is called plaintext. The protected data is called cipher text. Encryption is a procedure to convert plaintext into cipher text. Decryption is a procedure to convert cipher text into plaintext. A cryptographic system typically consists of algorithms, keys, and key management facilities. There are two basic types of cryptographic systems: symmetric ("private key") and asymmetric ("public key").

Symmetric key systems require both the sender and the recipient to have the same key. This key is used by the sender to encrypt the data, and again by the recipient to decrypt the data. Key exchange is clearly a problem. How do you securely send a key that will enable you to send other data securely? If a private key is intercepted or stolen, the adversary can act

as either party and view all data and communications. You can think of the symmetric crypto system as akin to the Chubb type of door locks. You must be in possession of a key to both open and lock the door. Asymmetric cryptographic systems are considered much more flexible. Each user has both a public key and a private key.

Messages are encrypted with one key and can be decrypted only by the other key. The public key can be published widely while the private key is kept secret. If Alice wishes to send Bob a secret, she finds and verifies Bob's public key, encrypts her message with it, and mails it off to Bob. When Bob gets the message, he uses his private key to decrypt it. Verification of public keys is an important step. Failure to verify that the public key really does belong to Bob leaves open the possibility that Alice is using a key whose associated private key is in the hands of an enemy. Public Key Infrastructures or PKI's deal with this problem by providing certification authorities that sign keys by a supposedly trusted party and make them available for download or verification. Asymmetric ciphers are much slower than their symmetric counterparts and key sizes are generally much larger. You can think of a public key system as akin to a Yale type door lock. Anyone can push the door locked, but you must be in possession of the correct key to open the door.

The project is devoted the description of cryptographic algorithms, particularly RSA algorithm over network. The Goal of RSA Algorithm is to implement a demonstrable application that will perform the encryption and decryption of a text file using RSA Algorithm. I will give input as plaintext and it will generate the corresponding ciphertext. Ciphertext is decrypted to get the original plain text.

R.S.A. stands for Rivest, Shamir and Adleman - the three cryptographers who invented the first practical commercial public key cryptosystem. Today it is used in web browsers, email programs, mobile phones, virtual private networks, secure shells, and many other places. Exactly how much security it provides is debatable, but with sufficiently large keys you can be confident of foiling the vast majority of attackers. Until recently the use of RSA was very much restricted by patent and export laws. However, the patent has now expired and US export laws have been relaxed.

CHAPTER ONE

1. OVERVIEW OF CRYPTOGRAPHY SYSTEMS

1.1 Introduction

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 mentioned.

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 fsecurity 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 What Does Cryptography mean

Cryptography means 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.

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.

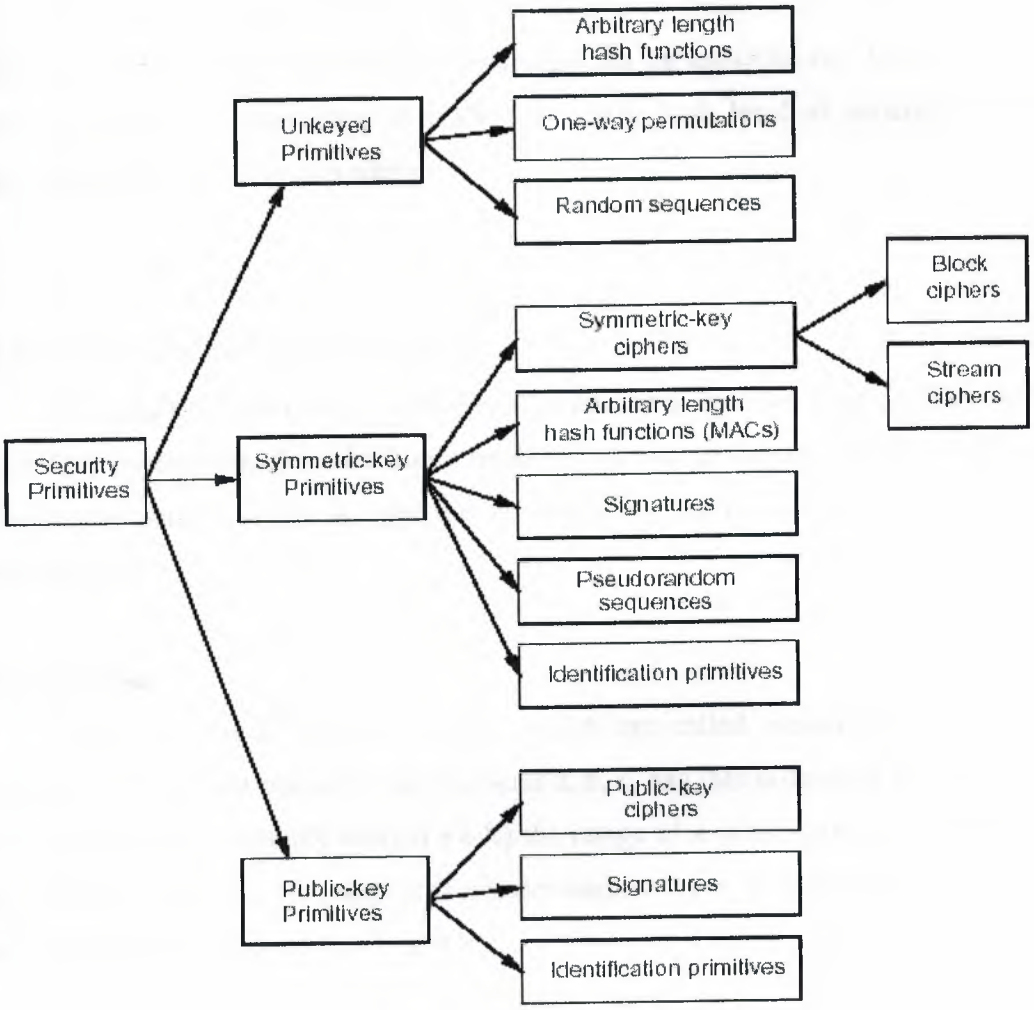

Figure 1.1 A taxonomy of cryptographic primitives.

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→Y.

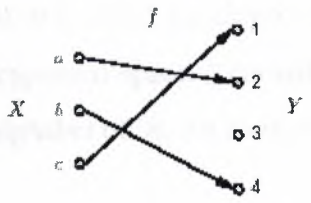

Figure 1.2 A function f from a set X to a set Y.

- 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→Y is 1-1 and Im (f) = Y, then f is called a bijection.

- One-way functions: A function f from a set X to a set Y is called a one-way function if f (x) is easy to compute for all $x \in X$ but for essentially all elements $y \in$ 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→S).
- Involutions: Involutions have the property that they are their own inverses. (i.e., f: S→ 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.
- C denotes a set called the cypertext space. C consists of strings of symbols from an alphabet; differ from the alphabet of $\mathcal{M}$. An element of C is called a cypertext.

1.3.2.2 Encryption and Decryption Transformations

- $\mathcal{K}$ 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 C, denoted by $\mathcal{E}e$.
- $\mathcal{D}_d$ denotes a bijection from 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 cypertext 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 ∈ M to Bob, she computes c = Ee (m) and transmits this to Bob. Upon receiving c, Bob computes D_d (c) = m and hence recovers the original message m.

The question arises as to why keys are necessary. If some particular encryption/decryption transformation is exposed then one does not have to redesign the entire scheme but simply change the key. Figure 1.3 provides a simple model of a two-party communication using encryption.

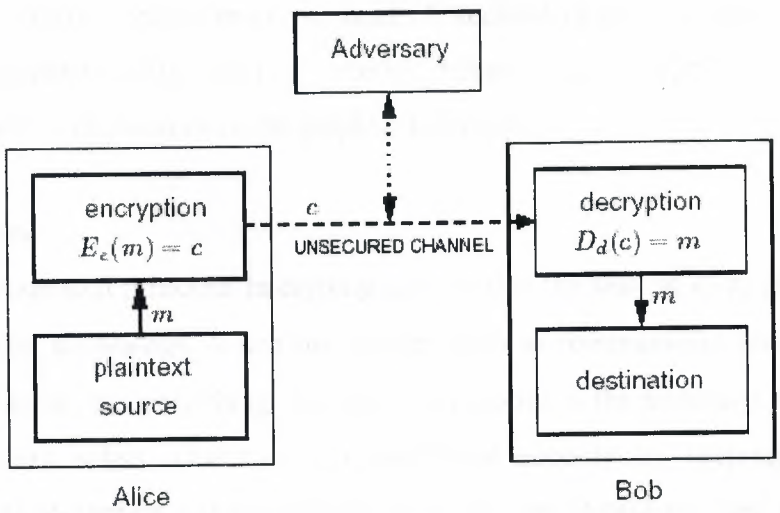

Figure 1.3 Schematic of a two-party communication.

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 $\mathcal{M}$; 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 cypertext within some appropriate time frame.

Trying all possible keys to see which one the communicating parties are using can break an encryption scheme. 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.

1.4 Symmetric-key Encryption

Consider an encryption scheme consisting of the sets of encryption and decryption transformations $\{\mathcal{E}e: e \in \mathcal{K}\}$ and $\{\mathcal{D}_d : d \in \mathcal{K}\}$, 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.

The block diagram of Figure 1.4, with the addition of the secure channel, can describe a two-party communication using symmetric-key encryption.

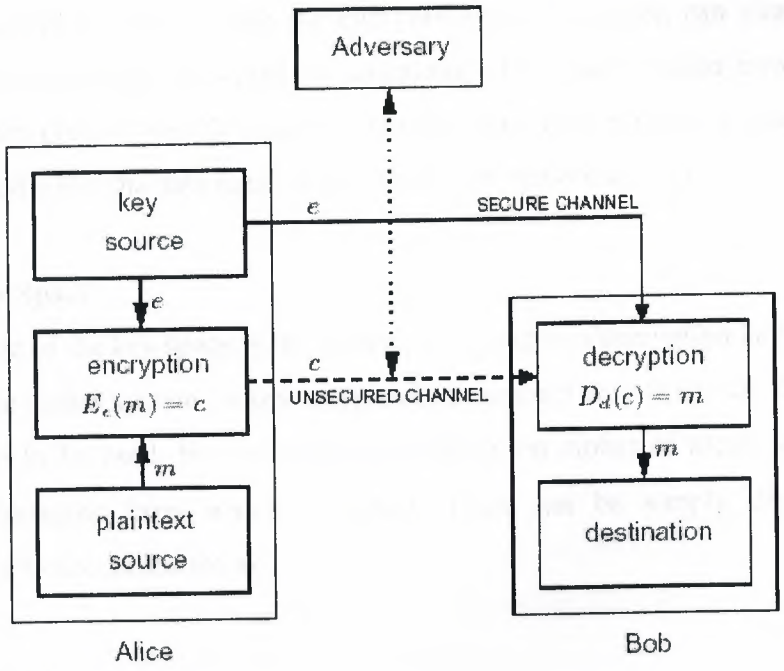

Figure 1.4 Two-party communication using encryption, with a secure channel

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 $S_{\mathcal{A}}$ and $V_{\mathcal{A}}$ provide a digital signature scheme for $\mathcal{A}$.

- $\mathcal{M}$ 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.
- $S_{\mathcal{A}}$ is a transformation from the message set $\mathcal{M}$ to the signature set $\mathcal{S}$, and is called a signing transformation for entity $\mathcal{A}$.

- $\mathcal{V}_A$ is a transformation from the set $\mathcal{M} \times S$ to the set {true, false} $\mathcal{V}_A$ 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 $\{\mathcal{E}e: e \in \mathcal{K}\}$ be a set of encryption transformations, and let $\{\mathcal{D}_d: d \in \mathcal{K}\}$ be the set of corresponding decryption transformations, where $\mathcal{K}$ is the key space. Consider any pair of associated encryption/decryption transformations $(\mathcal{E}e; \mathcal{D}d)$ and suppose that each pair has the property that knowing $\mathcal{E}e$ it is computationally infeasible, given a random ciphertext $c \in C$, to find the message $m \in \mathcal{M}$ such that $\mathcal{E}e(m) = c$. This property implies that given e it is infeasible to determine the corresponding decryption key d. $\mathcal{E}e$ 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

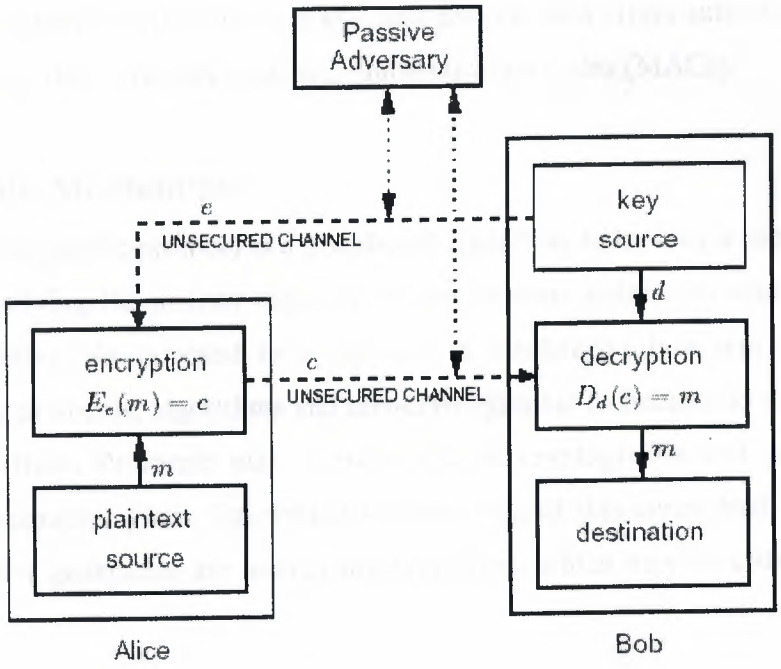

Figure 1.5 Encryption using public-key techniques.

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.

CHAPTER TWO

2. CRYPTOGRAPHY FUNCTIONS

2.1 Overview

In this chapter basic functions involved in cryptography are explained. Functions that 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.

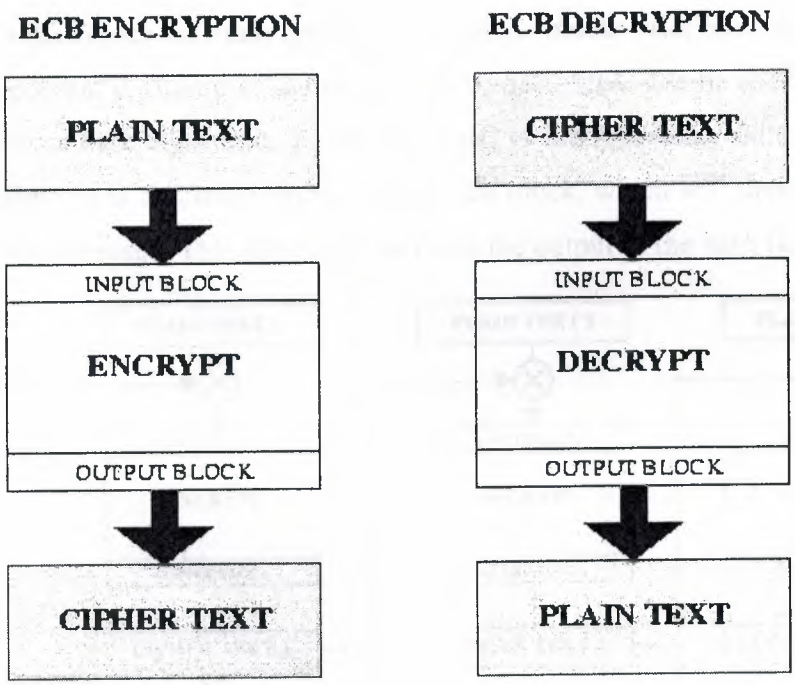

Figure 2.1: Shows a ECB Encryption/Decryption Model

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.

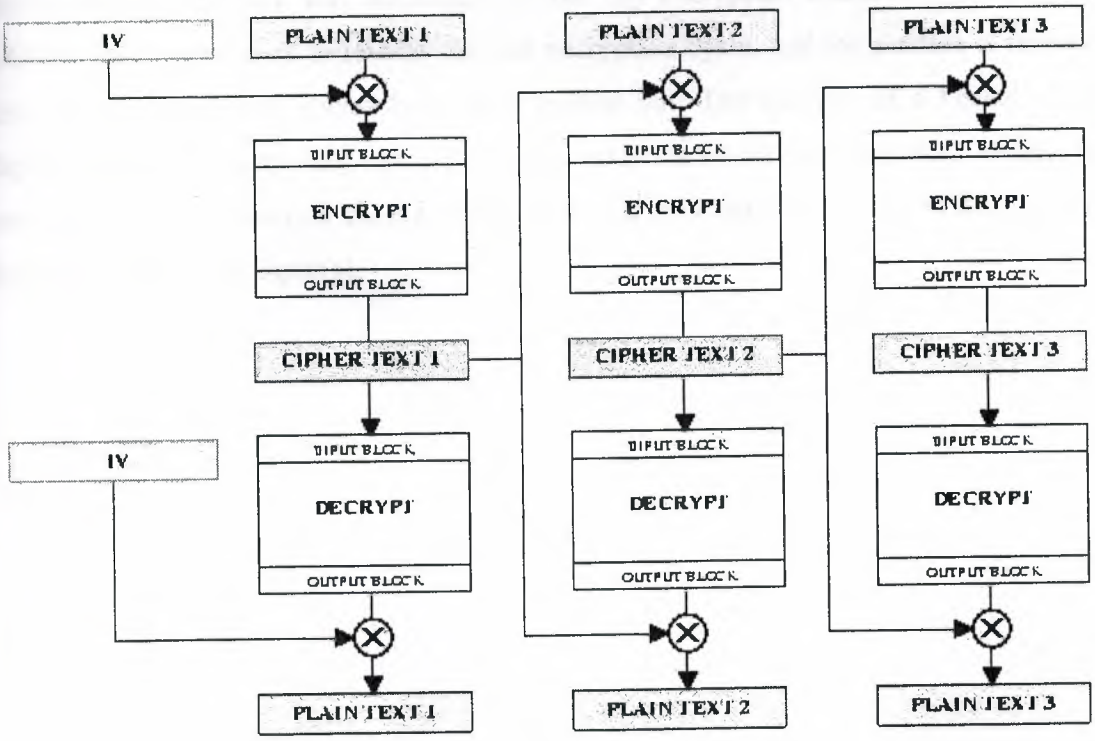

Figure 2.2: Shows a CBC Encryption/Decryption Model

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).

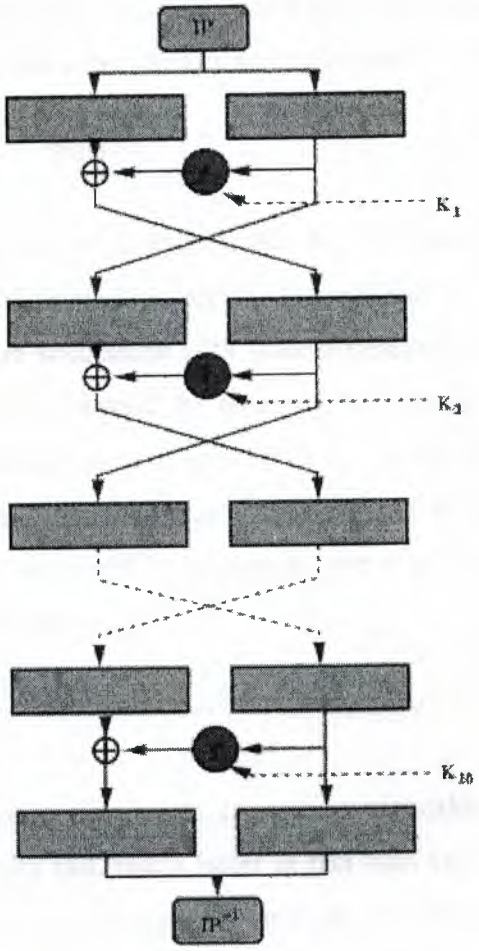

Figure 2.3: Shows a Feistel Model

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 combining the keystream with the plaintext, usually with the bitwise XOR operation, provides encryption. 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.

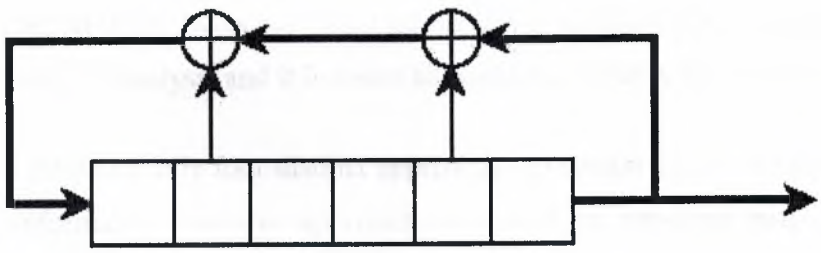

Figure 2.1: Shows a Linear Feed Back Register Model

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

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

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→B: h(h(K))
S6.	Bob verifies h(h(K)) is correct,	B→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:	$Q_A = g^{R}{}_A \bmod p$,	$A \rightarrow B$: $E_S (Q_A)$.
D2.	Bob computes:	$Q_B = g^{R}{}_B \bmod p$,	$B \rightarrow A$: $E_S (Q_B)$.
D3.	Alice computes:	$K = h(Q_B{}^{R}{}_A \bmod p)$	
D4.	Bob computes:	$K = h(Q_A{}^{R}{}_B \bmod p)$	

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 for 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 ($g^R{}_X$ mod 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:

$$K' = (E_{Si}^{-1}(E_S(g^{R}{}_B)))^{RA}$$

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:

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

$$= 2^{(S R_A)(X/S')} \bmod p$$

$$= 2^{(X R_A S/S')} \bmod p$$

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

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

Thus, when S' = S, K' = K. More generally, the attack works because the dictionary of passwords $\{S_1, S_2 \ldots S_n\}$ is equivalent to a dictionary of exponents $E = \{e_1, e_2 \ldots e_n\}$, 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 gc 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.

CHAPTER THREE

3. ENCRYPTION & DECRYPTION USING RSA ALGORITHM

3.1 Overview

RSA is a public-key cryptosystem developed by MIT professors: Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman in 1977 in an effort to help ensure Internet security. As Steve Burnett of RSA Data Security, Inc. described it, a cryptosystem is simply an algorithm that can convert input data into something unrecognizable (encryption), and convert the unrecognizable data back to its original form (decryption).

The RSA scheme is a block cipher in which the plaintext and cipher text are integers between 0 and n-1 for some n. A typical size for n is 1024 bits, or 309 decimal digits.

To encrypt data, enter the data ("plaintext") and an encryption key to the encryption portion of the algorithm. To decrypt the "cipher text," a proper decryption key is used at the decryption portion of the algorithm. Those keys, which contain simply a string of numbers, are called public key and private key, respectively. For example, suppose Alice intends to send e-mail to Bob. Through a public-key directory, she finds his public key. Then, she encrypts her message using the key and sends it to Bob. This public key, however, will not decrypt the cipher text. Knowledge of Bob's public key will not help an eavesdropper. In order for Bob to decrypt his cipher text, he must use his private key. If Bob wants to respond to Alice, he encrypts his message using her public key.

The challenge of public-key cryptography is developing a system in which it is impossible to determine the private key. This is accomplished through the use of a one-way function. With a one-way function, it is relatively easy to compute a result given some input values. In mathematical terms, given x, computing f(x) is easy, but given f(x), computing x is nearly impossible. The one-way function used in RSA is multiplication of prime numbers. It is easy to multiply two big prime numbers, but for most very large

primes, it is extremely time-consuming to factor them. Public-key cryptography uses this function by building a cryptosystem that uses two large primes to build the private key and the product of those primes to build the public key.

3.2 How does cryptographic algorithm work?

A *cryptographic algorithm,* or cipher, is a mathematical function used in the encryption and decryption process. A cryptographic algorithm works in combination with a *key* — a word, number, or phrase to encrypt the plaintext. The same plaintext encrypts to different cipher text with different keys. The security of encrypted data is entirely dependent on two things: the strength of the cryptographic algorithm and the secrecy of the key. A cryptographic algorithm, plus all possible keys and all the protocols that make it work comprise a *cryptosystem.* PGP is a cryptosystem.

3.3 Different types of Cryptosystems/Encryptions:

There are four ways of encryption that we mostly use in network system

- Pretty Good Privacy, PGP
- Rivest, Shamir, Adleman, RSA
- Keyless Encryption, KE
- One-Time Pads, OTP

3.3.1 PGP (Pretty Good Privacy)

Pretty Good Privacy. Up to this point we've talked about private-key cryptography (one key used y both parties). There was one problem with this kind of encryption: If the key was intercepted, a third party could decrypt the messages. So, the ideas of public-key cryptography were developed. Here's how it works...

Everyone has two keys: a public and a private key. When someone wants to send something to a recipient, they (the sender) encrypt it with the recipient's public key. Then the only way to decrypt it is with the recipient's private key. One of the other benefits to PGP is that it allows the sender to "sign" their messages. This proves that the message came from the sender and has not been altered in transport. Based on this theory, PGP

allows everyone to publicize their public keys, while keeping their private keys secret. The result is that anyone can encrypt a message to someone else, as long as they have that person's public key.

In actuality, PGP uses a series of private key, public key and one-way hash functions to encrypt a message. A one-way hash function takes some plaintext and translates it into a specific hash. The hash is unique to the message (like a fingerprint is to a person). The hash is also non-reversible, hence the name one-way. Let's run through an example of what PGP does to encrypt and decrypt an e-mail message. Our sender will be Chris and our receiver will be Brian.

- Chris writes his message.
- Chris uses a one-way hash function (such as MD5) to create a hash for the message.
- Chris, via RSA or some other digital signature algorithm, signs the hash with his private key.
- Chris merges the message and the signature, resulting in a new-signed message.
- A random encryption key is generated, the session key.
- Chris uses the session key to encrypt the message, using DES or some other private key method.
- Chris gets Brian's public key.
- Chris then encrypts the key with Brian's public key, via RSA or some other public key method.
- Chris merges the encrypted message and the encrypted key and mails it to Brian.

Once Brian receives the message he can have PGP decrypt it. Here's what it would do:

- Brian separates the encrypted message and the encrypted session key.
- Using RSA, Brian decrypts the session key.
- Using DES, Brian decrypts the message with the decrypted session key.
- Brian then separates the message and the signature.

- Using MD5, Brian calculates the hash value of the message.
- Brian gets Chris' public key.
- Via RSA, and Chris' public key, Brian decrypts the signature.
- Brian then compares the hash value and the decrypted signature. If they are the same, Brian knows that the message is authentic and has not been altered since Chris signed it.
-

3.3.2 RSA (Rivest, Shamir, Adleman)

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

$$C = M^e \bmod n$$

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

$$M = C^d \bmod n$$

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

values through equation (2) and get 19, our original plaintext! Try it yourself with other numbers.

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

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

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

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

3.4 The RSA Algorithm

Let's run the RSA algorithm through steps of how RSA algorithm procedure does work:

3.4.1 Key Generation

1.Generate two large prime numbers, p and q

2. Let n = pq

3. Let m = (p-1)(q-1)

4. Choose a small number e, coprime to m

5. Find d, such that de % m = 1

Publish e and n as the public key, Keep d and n as the secret k

Encryption

$C = P^e \% n$

Decryption

$P = C^d \% n$

x % y means the remainder of x divided by y

The reasons why this algorithm works are discussed in the mathematics section. Its security comes from the computational difficulty of factoring large numbers. To be secure, very large numbers must be used for p and q - 100 decimal digits at the very least.

I'll now go through a simple worked **example 1.**

a) Key Generation

1) Generate two large prime numbers, p and q

To make the example easy to follow I am going to use small numbers, but this is not secure. To find random primes, we start at a random number and go up ascending odd numbers until we find a prime. Lets have:

p = 7
q = 19

2) Let n = pq
n = 7 * 19
= 133

3) Let m = (p - 1)(q - 1)
m = (7 - 1)(19 - 1)
= 6 * 18
= 108

4) Choose a small number, e coprime to m

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

e = 2 => gcd(e, 108) = 2 (no)

e = 3 => gcd(e, 108) = 3 (no)

e = 4 => gcd(e, 108) = 4 (no)

e = 5 => gcd(e, 108) = 1 (yes!)

5) Find d, such that de % m = 1

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

n = 0 => d = 1 / 5 (no)

n = 1 => d = 109 / 5 (no)

n = 2 => d = 217 / 5 (no)

n = 3 => d = 325 / 5

= 65 (yes!)

To do this with big numbers, a more sophisticated algorithm called extended Euclid must be used.

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

b) Encryption

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

$C = P^e \% n$
$= 6^5 \% 133$
$= 7776 \% 133$
$= 62$

c) Decryption

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

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

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

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

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

3.5 From Applied Cryptography

$n = pq$

$ed \_ 1 \bmod (p \_ 1)(q \_ 1)$

$c = me \bmod n$

$m = cd \bmod n$

n is the product of p and q, two prime numbers.

The product of e and d divided by (p_1)(q _1) has a remainder of 1. C is the remainder of me divided by n; m is the remainder of cd divided by n.

3.5.1 The Product of Two Primes

n is the product of two prime numbers, p and q. n is made public as it is used in decryption as well as encryption. p and q should never be revealed.

Example:

p = 59, q = 67; n = 59 _ 67 = 3953

3.5.2 e and d, The Keys

e, the encryption key, is a random number relatively prime to (p _ 1)(q _ 1). d can be calculated to be de = 1 + x(p _ 1)(q _ 1), x is a throwaway value as it is \lost" in the modulus calculation. e is the public key and d is the private key.

Example, continued:

e = 7, (p _ 1)(q _ 1) = 3828

de = 1 + x(p _ 1)(q _ 1)

xd = 547, x = 1, d = 547

e = 7, d = 547

3.6 Key types

Two key types are employed in the primitives and schemes defined: *RSA public key* and *RSA private key*. Together, an RSA public key and an RSA private key form an *RSA key pair*.

This specification supports so-called "multi-prime" RSA where the modulus may have more than two prime factors. Better performance can be achieved on single processor platforms, but to a greater extent on multiprocessor platforms, where the modular exponentiations involved can be done in parallel.

3.6.1 RSA public key

RSA public key consists of two components:

n the RSA modulus, a positive integer

e the RSA public exponent, a positive integer

In a *valid RSA public key*, the RSA modulus n is a product of u distinct odd primes r_i, $i = 1, 2, \ldots, u$, where $u \geq 2$, and the RSA public exponent e is an integer between 3 and $n - 1$ satisfying GCD $(e, \lambda(n)) = 1$, where $\lambda(n) = \text{LCM}(r_1 - 1, \ldots, r_u - 1)$. By convention, the first two primes r_1 and r_2 may also be denoted p and q respectively.

3.6.2 RSA private key

An RSA private key may have either of two representations.

1. The first representation consists of the pair (n, d), where the components have the following meanings:

n the RSA modulus, a positive integer

d the RSA private exponent, a positive integer

2. The second representation consists of a quintuple $(p, q, dP, dQ, qInv)$ and a (possibly empty) sequence of triplets (r_i, d_i, t_i), $i = 3, \ldots, u$, one for each prime not in the quintuple, where the components have the following meanings:

p	the first factor, a positive integer
q	the second factor, a positive integer
dP	the first factor's CRT exponent, a positive integer
dQ	the second factor's CRT exponent, a positive integer
$qInv$	the (first) CRT coefficient, a positive integer
r_i	the i^{th} factor, a positive integer
d_i	the i^{th} factor's CRT exponent, a positive integer
t_i	the i^{th} factor's CRT coefficient, a positive integer

In a *valid RSA private key* with the first representation, the RSA modulus n is the same as in the corresponding RSA public key and is the product of u distinct odd primes

ri, i = 1, 2, …, u, where u ≥ 2. The RSA private exponent d is a positive integer less than n satisfying

$$e \cdot d \equiv 1 \pmod{\lambda(n)},$$

Where e is the corresponding RSA public exponent and $\lambda(n)$ is defined before.

In a valid RSA private key with the second representation, the two factors p and q are the *first two* prime factors of the RSA modulus n (i.e., r_1 and r_2), the CRT exponents dP and dQ are positive integers less than p and q respectively satisfying

$$e \cdot dP \equiv 1 \pmod{(p-1)}$$
$$e \cdot dQ \equiv 1 \pmod{(q-1)},$$

And the CRT coefficient $qInv$ is a positive integer less than p satisfying

$$q \cdot qInv \equiv 1 \pmod{p}.$$

If $u > 2$, the representation will include one or more triplets (r_i, d_i, t_i), $i = 3, \ldots, u$. The factors r_i are the additional prime factors of the RSA modulus n. Each CRT exponent d_i ($i = 3, \ldots, u$) satisfies

$$e \cdot d_i \equiv 1 \pmod{(r_i - 1)}.$$

Each CRT coefficient t_i ($i = 3, \ldots, u$) is a positive integer less than r_i satisfying

$$R_i \cdot t_i \equiv 1 \pmod{r_i},$$

Where $R_i = r_1 \cdot r_2 \cdot \ldots \cdot r_{i-1}$.

3.7 How fast is the RSA algorithm

An "RSA operation," whether encrypting, decrypting, signing, or verifying is essentially a modular exponentiation. This computation is performed by a series of modular multiplications.

In practical applications, it is common to choose a small public exponent for the public key. In fact, entire groups of users can use the same public exponent, each with a different modulus. (There are some restrictions on the prime factors of the modulus when the public exponent is fixed.) This makes encryption faster than decryption and verification faster than signing. With the typical modular exponentiation algorithms used to implement the RSA algorithm, public key operations take $O(k^2)$ steps, private key

operations take $O(k^3)$ steps, and key generation takes $O(k^4)$ steps, where k is the number of bits in the modulus. ``Fast multiplication'' techniques, such as methods based on the Fast Fourier Transform (FFT), require asymptotically fewer steps. In practice, however, they are not as common due to their greater software complexity and the fact that they may actually be slower for typical key sizes.

The speed and efficiency of the many commercially available software and hardware implementations of the RSA algorithm are increasing rapidly

By comparison, DES and other block ciphers are much faster than the RSA algorithm. DES is generally at least 100 times as fast in software and between 1,000 and 10,000 times as fast in hardware, depending on the implementation. Implementations of the RSA algorithm will probably narrow the gap a bit in coming years, due to high demand, but block ciphers will get faster as well.

3.8 Encryption and Decryption

The encryption and decryption math is more Straightforward, yet more demanding. The remainder of the message, or message block, value multiplied by itself key times divided by the modulus n is the encrypted or decrypted value, depending on which key is used. The value of the message, or message block, must be less than the value of n.

Example, continued:
Message: 0920 2000
09207 mod 3953 = 0307
20007 mod 3953 = 2497
0307547 mod 3953 = 0920
2497547 mod 3953 = 2000

3.8.1 The Mathematical Guts of RSA Encryption

Here's the algorithm behind RSA public key encryption:

Find *P* and *Q*, two large (1024-bit) prime numbers.
Choose *E* such that *E* is greater than 1, *E* is less than *PQ*, and *E* and *(P-1)(Q-1)* are *relatively prime*, which means they have no prime factors in common. *E* does not have to be prime, but it must be odd. *(P-1)(Q-1)* can't be prime because it's an even number.
Compute *D* such that *(DE - 1)* is evenly divisible by *(P-1)(Q-1)*. Mathematicians write this as $DE \equiv 1 \pmod{(P-1)(Q-1)}$, and they call *D* the *multiplicative inverse* of *E*. This is easy to do -- simply find an integer X which causes *D = (X(P-1)(Q-1) + 1)/E* to be an integer, then use that value of *D*.

The encryption function is *C = (T^E) mod PQ*, where *C* is the ciphertext (a positive integer), *T* is the plaintext (a positive integer), and ^ indicates exponentiation. The message being encrypted, *T* must be less than the modulus, *PQ*.

The decryption function is *T = (C^D) mod PQ*, where *C* is the ciphertext (a positive integer), *T* is the plaintext (a positive integer), and ^ indicates exponentiation.

The *public key* is the pair *(PQ, E)*. The *private key* is the number *D* (reveal it to no one). The product *PQ* is the *modulus* (often called N in the literature). *E* is the *public exponent*. *D* is the *secret exponent*.

The public key can be published freely, because there are no known easy methods of calculating *D*, *P*, or *Q* given only *(PQ, E)* (your public key). If *P* and *Q* were each 1024 bits long, it would be millions of years before the most powerful computers presently in existence can factor your modulus into *P* and *Q*.

3.8.2 RSA public-key encryption

The RSA cryptosystem, named after its inventors R. Rivest, A. Shamir, and L. Adleman, is the most widely used public-key cryptosystem. It may be used to provide both secrecy and digital signatures and its security is based on the intractability of the integer factorization.

3.8.2.1 Algorithm Key generations for RSA public-key encryption

Each entity creates an RSA public key and a corresponding private key.

Each entity A should do the following:

1. Generate two large random (and distinct) primes p and q, each roughly the same size.

2. Compute n = pq and _ = (p − 1)(q − 1).

3. Select a random integer e, 1 < e < _, such that gcd(e; _) = 1.

4. Use the extended Euclidean algorithm to compute the unique integer d, 1 < d < _, such that ed _ 1 (mod _).

5. A's public key is (n; e); A's private key is d.

Definition

The integers e and d in RSA key generation are called the *encryption exponent* and the *decryption exponent*, respectively, while n is called the *modulus*.

3.8.2.2 Algorithm RSA public-key encryption

B encrypts a message m for A, which A decrypts.

1. *Encryption.* B should do the following:

(a) Obtain A's authentic public key (n; e).

(b) Represent the message as an integer m in the interval [0; n − 1].

(c) Compute c = me mod n

(d) Send the cipher text c to A.

2. *Decryption.* To recover plaintext m from c, A should do the following:

(a) Use the private key d to recover m = cd mod n

Proof that decryption works. Since ed _ 1 (mod _), there exists an integer k such that ed = 1+k_. Now, if gcd(m; p) = 1 then by Fermat's theorem m^{p-1} _1 (modp):
Raising both sides of this congruence to the power k(q−1) and then multiplying both sides by m yields $m^{1+k(p-1)(q-1)}$ _ m (mod p):
On the other hand, if gcd(m; p) = p, then this last congruence is again valid since each side is congruent to 0 modulo p. Hence, in all cases m^{ed} _ m (mod p):

By the same argument, med _ m (mod q):

Finally, since p and q are distinct primes, it follows that

med _ m (mod n) and, hence,

cd _ (me)d _ m (mod n):

3.8.2.3 RSA encryption with artificially small parameter's example

Key generation. Entity A chooses the primes p = 2357, q = 2551, and computes n = pq = 6012707 and _ = (p−1)(q−1) = 6007800. A chooses e = 3674911 and, using the extended Euclidean algorithm, finds d = 422191 such that ed _ 1 (mod _). A's public key is the pair (n = 6012707; e = 3674911), while A's private key is d = 422191. *Encryption.* To encrypt a message m = 5234673, B uses an algorithm for modular exponentiation to compute:

c = me mod n = 52346733674911 mod 6012707 = 3650502; and sends this to A.

Decryption. To decrypt c, A computes

cd mod n = 3650502422191 mod 6012707 = 5234673.

3.8.2.4 Universal exponent

The number _ = lcm(p−1; q −1), sometimes called the *universal exponent* of n, may be used instead of _ = (p − 1)(q − 1) in RSA key generation Observe that _ is a proper divisor of _. Using _ can result in a smaller decryption exponent d, which may result in faster decryption However, if p and q are chosen at random, then gcd(p−1; q−1) is expected to be small, and consequently _ and _ will be roughly of the same size

3.9 Encryption Program

This program encrypted the plain text file of "text.txt" (Appendix) into the cyphertext file of "out.txt" (Appendix), which consisted of integer stream. At the same time, the public key pair and private key were generated and saved in the file of "key.txt" (Appendix). Finally, the decryption algorithm decrypted the cyphertext file of "out.txt"

into the file of "out_d.txt" (Appendix) with the aid of the private key pair and the private key.

In the main() function of "project_rsa.c", plaintext file was translated into ASCII code first as a stream of integers, and saved in the temporary file of "out". In order to hide the occurrence frequency of letters in English text, the letter was not encrypted respectively using RSA Encryption method. Instead, we extracted number with the fixed digits from the integer stream.

For example, using the key values in "project_rsa.c", the plaintext of "A3D4" can be translated as decimal integers stream of "65516852" according to the ASCII code. If we chose the 3 fixed digits, then this decimal integer stream can be separated as 655, 168, 52. It is noted that the last number should not be a 3 digits number. As such, we add '0' at the end of the number to expend as 520. Finally, the decimal integer stream can be separated as 655, 168, and 520. These three numbers were used for the encryption using RSA method. Accordingly, the cyphertext would be generated as: 28900 23450 20606 with the public key pair of (49447, 3513) and the private key of "11577".

Finally, the program read the cyphertext from the file of "out.txt". The original decimal integer stream can be decrypted using theses public and private keys, and save in the temporary file of "out_d". According to the ASCII code, all the English letters have ASCII code larger than 31 (We included the SPACE key) and less than 125. Then the plaintext can be easily translated from the original decimal integer stream. The decrypted plaintext was save in the file of "out_d.txt".

In order to access the program, the access process was inserted in the Encryption process in the main () function in "project_rsa.c".

Unlike other algorithm, the RSA test vector is so hard to find. To test this algorithm, and the implementation is in "rsa_test.c" (replace the main function of project_rsa.c), in which the key values are as following, public key = (3233, 17), private key = 2753. The plaintext value of 123 can be encrypted as 855.

3.9.1 Program List

Source Files:

project_rsa.c	main program
exponentiation.c	carry out the calculation of exponentiation
keys.c	generate the public and private keys
prime.c	find the primes use in the key generation
gcd.c	find the GCD of a and b
inverse.c	An implementation of the extended Euclidean algorithm
rsa_test.c	This is used for the test of the algorithm.

Header Files:

exponentiation.h

keys.h

gcd.h

inverse.h

3.10 RSA and related signature schemes

This section describes the RSA signature scheme and other closely related methods. The security of the schemes presented here relies to a large degree on the intractability of the integer factorization problem; the schemes presented include both digital signatures with message recovery and appendix.

3.10.1 The RSA signature scheme

The message space and ciphertext space for the RSA public-key encryption scheme are both Zn = f0; 1; 2; : : : ; n − 1g where n = pq is the product of two randomly chosen distinct prime numbers. Since the encryption transformation is a bijection, digital signatures can be created by reversing the roles of encryption and decryption. The RSA signature scheme is a deterministic digital signature scheme, which provides message

recovery .The signing spaceMS and signature space S, are both Zn. A redundancy function R: M−!Zn is chosen and is public knowledge.

3.10.1.1 Algorithm Key generation for the RSA signature scheme

Each entity creates an RSA public key and a corresponding private key.

Each entity A should do the following:

1. Generate two large distinct random primes p and q, each roughly the same size

2. Compute n = pq and _ = (p − 1)(q − 1).

3. Select a random integer e, 1 < e < _, such that gcd(e; _) = 1.

4. Use the extended Euclidean algorithm to compute the unique integer d, 1 < d < _, such that ed _ 1 (mod _).

5. A's public key is (n; e); A's private key is d.

3.10.1.2 Algorithm RSA signature generation and verification

Entity A signs a messagem2M. Any entity B can verify A's signature and recover the message m from the signature.

1. *Signature generation.* Entity A should do the following:

(a) Compute em = R(m), an integer in the range [0; n − 1].

(b) Compute s = emd mod n.

(c) A's signature for m is s.

2. *Verification.* To verify A's signature s and recover the message m, B should:

(a) Obtain A's authentic public key (n; e).

(b) Compute em = se mod n.

(c) Verify that em2MR; if not, reject the signature.

(d) Recover $m = R^{-1}(\bar{m})$.

Proof that signature verification works. If s is a signature for a message m, then $s \equiv \bar{m}^d \bmod n$ where $\bar{m} = R(m)$. Since $ed \equiv 1 \pmod{\phi}$, $s^e \equiv \bar{m}^{ed} \equiv \bar{m} \pmod{n}$. Finally, $R^{-1}(\bar{m}) = R^{-1}(R(m)) = m$.

3.10.2 Possible attacks on RSA signatures

(i) Integer factorization

If an adversary is able to factor the public modulus n of some entity A, then the adversary can compute _ and then, using the extended Euclidean algorithm, dedu the private key d from _ and the public exponent e by solving ed _ 1 (mod _). This constitutes a total break of the system. To guard against this, A must select p and q so that factoring n is a computationally infeasible task. For further information.

(ii) Multiplicative property of RSA

The RSA signature scheme (as well as the encryption method, cf has the following multiplicative property, sometimes referred to as the *homomorphic property*. If s1 = md1 mod n and s2 = md2 mod n are signatures on messages m1 and m2, respectively (or more properly on messages with redundancy added), then s = s1s2 mod n has the property that s = (m1m2)d mod n. Ifm = m1m2 has the proper redundancy (i.e.,m2MR), then s will be a valid signature for it.

Hence, it is important that the redundancy function R is not multiplicative, i.e., for essentially all pairs a; b 2 M, R(a _ b) 6= R(a)R(b). As

3.11 Security of RSA

This subsection discusses various security issues related to RSA encryption. Various attacks, which have been studied in the literature, are presented, as well as appropriate measures to counteract these threats.

(i) Relation to factoring

The task faced by a passive adversary is that of recovering plaintext from the corresponding ciphertext c, given the public information (n; e) of the intended receiver A. This is called the *RSA problem* (RSAP), which was introduced in x3.3. There is no efficient algorithm known for this problem.

One possible approach, which an adversary could employ to solving the RSA problem is to first factor n, and then compute e _ and d just as A did in Algorithm. Once d is obtained, the adversary can decrypt any ciphertext intended for A.

On the other hand, if an adversary could somehow compute d, then it could subsequently factor n efficiently as follows. First note that since ed _ 1 (mod _), there is an integer k such that ed − 1 = k_. Hence, by Fact 2.126(i), aed−1 _ 1 (mod n) for all a 2 Z_ n. Let ed − 1 = 2s t, where t is an odd integer. Then it can be shown that there exists an i 2 [1; s] such that a2i−1t 6_ _1 (mod n) and a2it _ 1 (mod n) for at least half of all a 2 Z_n; if a and i are such integers then gcd(a2i−1t − 1; n) is a non-trivial factor of n. Thus the adversary simply needs to repeatedly select random a 2 Z_ n and check if an i 2 [1; s] satisfying the above property exists; the expected number of trials before a non-trivial factor of n is obtained is 2. This discussion establishes the following.

Fact The problem of computing the RSA decryption exponent d from the public key (n; e), and the problem of factoring n, are computationally equivalent.

When generating RSA keys, it is imperative that the primes p and q be selected in such a way that factoring n = pq is computationally infeasible.

(ii) Small encryption exponent e.

In order to improve the efficiency of encryption, it is desirable to select a small encryption exponent e such as e = 3. A group of entities may all have the same encryption exponent e, however, each entity in the group must have its own distinct modulus (cf. x8.2.2(vi)). If an entity A wishes to send the same message m to three entities whose public moduli are n1, n2, n3, and whose encryption exponents are e = 3, then A would send ci = m3 mod ni, for i = 1; 2; 3. Since these moduli are most likely pairwise relatively

prime, an eavesdropper observing c1, c2, c3 can use Gauss's algorithm to find a solution x, 0 _ x < n1n2n3, to the three congruences 8<: x _ c1 (mod n1) x _ c2 (mod n2) x _ c3 (mod n3):

Since m3 < n1n2n3, by the Chinese remainder theorem it must be the case that x = m3. Hence, by computing the integer cube root of x, the eavesdropper can recover the plaintext m.

Thus a small encryption exponent such as e = 3 should not be used if the same message, or even the same message with known variations, is sent to many entities. Alternatively, to prevent against such an attack, a pseudorandomly generated bitstring of appropriate Length should be appended to the plaintext message prior to encryption; the pseudorandom bitstring should be independently generated for each encryption. This process is sometimes referred to as *salting* the message. Small encryption exponents are also a problem for small messages m, because ifm< n1=e, then m can be recovered from the ciphertext c = me mod n simply by computing the integer eth root of c; salting plaintext messages also circumvents this problem.

(iii) Forward search attack

If themessage space is small or predictable, an adversary can decrypt a ciphertext c by simply encrypting all possible plaintext messages until c is obtained. Salting the message as described above is one simple method of preventing such an attack.

(iv) Small decryption exponent d

As was the case with the encryption exponent e, it may seem desirable to select a small decryption exponent d in order to improve the efficiency of decryption.1 However, if gcd(p− 1; q − 1) is small, as is typically the case, and if d has up to approximately one-quarter as many bits as the modulus n, then there is an efficient algorithm for computing d from the public information (n; e). This algorithm cannot be extended to the case where d is approximately the same size as n. hence, to avoid this attack; the decryption exponent d should be roughly the same size as n.

(v) Multiplicative properties

Let m1 and m2 be two plaintext messages, and let c1 and c2 be their respective RSA encryptions. Observe that (m1m2)e _ me 1me 2 _ c1c2 (mod n):

In other words, the ciphertext corresponding to the plaintext m = m1m2 mod n is c = c1c2 mod n; this is sometimes referred to as the *homomorphic property* of RSA. This observation leads to the following *adaptive chosen-ciphertext attack* on RSA encryption.

Suppose that an active adversarywishes to decrypt a particular ciphertext c = me mod n intended for A. Suppose also that A will decrypt arbitrary ciphertext for the adversary, other than c itself. The adversary can conceal c by selecting a random integer x 2 Z_ n and computing c = cxe mod n. upon presentation of c, A will compute for the adversary

m = (c)d mod n. Since

m _ (c)d _ cd(xe)d _ mx (mod n);

The adversary can then compute m = mx−1 mod n.

This adaptive chosen-ciphertext attack should be circumvented in practice by imposing some structural constraints on plaintext messages. If a ciphertext c is decrypted to amessage not possessing this structure, then c is rejected by the decryptor as being fraudulent. Now, if a plaintext message m has this (carefully chosen) structure, then with high probability mx mod n will not for x 2 Z_ n. Thus the adaptive chosen-ciphertext attack described in the previous paragraph will fail because A will not decrypt c for the adversary. Provides a powerful technique for guarding against adaptive chosen-ciphertext and other kinds of attacks.

(vi) Common modulus attack

The following discussion demonstrates why it is imperative for each entity to choose its own RSA modulus n.

It is sometimes suggested that a central trusted authority should select a single RSA modulus n, and then distribute a distinct encryption/decryption exponent pair (ei; di) to each entity in a network. However, as shown in (i) above, knowledge of any (ei; di) pair allows for the factorization of the modulus n, and hence any entity could subsequently determine the decryption exponents of all other entities in the network. Also, if a

singlemessage were encrypted and sent to two or more entities in the network, then there is a technique by which an eavesdropper (any entity not in the network) could recover the message with high probability using only publicly available information.

(vii) Cycling attacks

Let c = me mod n be a ciphertext. Let k be a positive integer such that cek _ c (mod n); since encryption is a permutation on the message space f0; 1; : : : ; n − 1g such an integer

k must exist. For the same reason it must be the case that cek−1 _ m (mod n). This observation leads to the following *cycling attack* on RSA encryption. An adversary computes ce mod n, ce2 mod n, ce3 mod n; : : : until c is obtained for the first time. If cek mod n = c, then the previous number in the cycle, namely cek−1 mod n, is equal to the plaintext m.

A *generalized cycling attack* is to find the smallest positive integer u such that f = gcd(ceu − c; n) > 1. If ceu _ c (mod p) and ceu 6_ c (mod q) then f = p. Similarly, if ceu 6_ c (mod p) and ceu _ c (mod q) (8.2) then f = q. In either case, n has been factored, and the adversary can recover d and then m. On the other hand, if both ceu _ c (mod p) and ceu _ c (mod q); then f = n and ceu _ c (mod n). In fact, u must be the smallest positive integer k for which cek _ c (mod n). In this case, the basic cycling attack has succeeded and so m = ceu−1 mod n can be computed efficiently. Since is expected to occur much less frequently than the generalized cycling attack usually terminates before the cycling attack does. For this reason, the generalized cycling attack can be viewed as being essentially an algorithm for factoring n.

Since factoring n is assumed to be intractable, these cycling attacks do not pose a threat to the security of RSA encryption.

(viii) Message concealing

A plaintext message m, 0 _ m _ n−1, in the RSA public-key encryption scheme is said to be *unconcealed* if it encrypts to itself; that is, me _ m (mod n). There are always

some messages, which are unconcealed (for example m = 0, m = 1, and m = n − 1). In fact, the number of unconcealed messages is exactly

$[1 + \gcd(e - 1; p - 1)] \_ [1 + \gcd(e - 1; q - 1)]$:

Since e−1, p−1 and q −1 are all even, the number of unconcealed messages is always at least 9. If p and q are random primes, and if e is chosen at random (or if e is chosen to be a small number such as e = 3 or e = 216 + 1 = 65537), then the proportion of messages which are unconcealed by RSA encryption will, in general, be negligibly small, and hence unconcealed messages do not pose a threat to the security of RSA encryption in practice.

3.12 RSA encryption in practice

There are numerous ways of speeding up RSA encryption and decryption in software and hardware implementations. Some of these techniques are covered, including fast modular multiplication (x14.3), fast modular exponentiation (x14.6), and the use of the Chinese remainder theorem for faster decryption, Even with these improvements, RSA encryption/decryption is substantially slower than the commonly used symmetric-key encryption algorithms such as DES

In practice, RSA encryption is most commonly used for the transport of symmetric-key encryption algorithm keys and for the encryption of small data items.

The RSA cryptosystem has been patented in the U.S. and Canada. Several standards organizations have written, or are in the process of writing, standards that address the use of the RSA cryptosystemfor encryption, digital signatures, and key establishment. For discussion of patent and standards issues related to RSA,

3.12.1 Recommended size of modulus

Given the latest progress in algorithms for factoring integers (x3.2), a 512-bit modulus n provides onlymarginal security from concerted attack.

As of 1996, in order to foil the powerful quadratic sieve (x3.2.6) and number field sieve

(x3.2.7) Factoring algorithms, a modulus n of at least 768 bits is recommended. For long-term security, 1024-bit or larger moduli should be used.

3.12.2 Selecting primes

(i) As mentioned in x8.2.2(i), the primes p and q should be selected so that factoring n = pq is computationally infeasible. The major restriction on p and q in order to avoid the elliptic curve factoring algorithm (x3.2.4) is that p and q should be about the same bitlength, and sufficiently large. For example, if a 1024-bit modulus n is to be used, then each of p and q should be about 512 bits in length.

(ii) Another restriction on the primes p and q is that the difference p − q should not be too small. If p − q is small, then p _ q and hence p _ p n. Thus, n could be factored efficiently simply by trial division by all odd integers close to p n. If p and q are chosen at random, then p − q will be appropriately large with overwhelming probability.

(iii) In addition to these restrictions, many authors have recommended that p and q be strong primes. A prime p is said to be a *strong prime* if the following three conditions are satisfied:

(a) p − 1 has a large prime factor, denoted r;

(b) p + 1 has a large prime factor; and

(c) r − 1 has a large prime factor.

An algorithm for generating strong primes is presented in x4.4.2. The reason for condition

(a) is to foil Pollard's p−1 factoring algorithm(x3.2.3) which is efficient only if n has a prime factor p such that p − 1 is smooth. Condition (b) foils the p + 1 factoring algorithm mentioned on page 125 in x3.12, which is efficient only if n has a prime factor p such that p + 1 is smooth. Finally, condition (c) ensures that the cycling attacks described in x8.2.2(vii) will fail. If the prime p is randomly chosen and is sufficiently

arge, then both p−1 and p+1 can be expected to have large prime factors. In any case, while strong primes protect against the p−1 and p+1 factoring algorithms, they do not protect against their generalization, the elliptic curve-factoring algorithm (x3.2.4). The latter is successful in factoring n if a randomly chosen number of the same size as p (more precisely, this number is the order of a randomly selected elliptic curve defined over Zp) has only small prime factors. Additionally, it has been shown that the chances of a cycling attack succeeding are negligible if p and q are randomly chosen (cf. x8.2.2 (vii)). Thus, strong primes offer little protection beyond that offered by random primes. Given the current state of knowledge of factoring algorithms, there is no compelling reason for requiring the use of strong primes in RSA key generation. On the other hand, they are no less secure than random primes, and require only minimal additional running time to compute; thus there is little real additional cost in using them.

3.12.3 Small encryption exponents

(i) If the encryption exponent e is chosen at random, then RSA encryption using the repeated square-and-multiply algorithm takes k modular squaring and an expected k=2 (less with optimizations) modular multiplications, where k is the bit length of the modulus n. Encryption can be sped up by selecting e to be small and/or by selecting e with a small number of 1's in its binary representation.

(ii) The encryption exponent e = 3 is commonly used in practice; in this case, it is necessary that neither p−1 nor q−1 be divisible by 3. This results in a very fast encryption operation since encryption only requires 1 modular multiplication and 1 modular squaring. Another encryption exponent used in practice is e = 216 + 1 = 65537.

This number has only two 1's in its binary representation, and so encryption using the repeated square-and-multiply algorithm requires only 16 modular squaring and 1 modular multiplication. The encryption exponent e = 216 + 1 has the advantage over e = 3 in that it resists the kind of attack discussed in x8.2.2(ii), since it is unlikely the same message will be sent to 216+1 recipients.

CHAPTER FOUR

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 cannot work without the services provided by the layer below it.

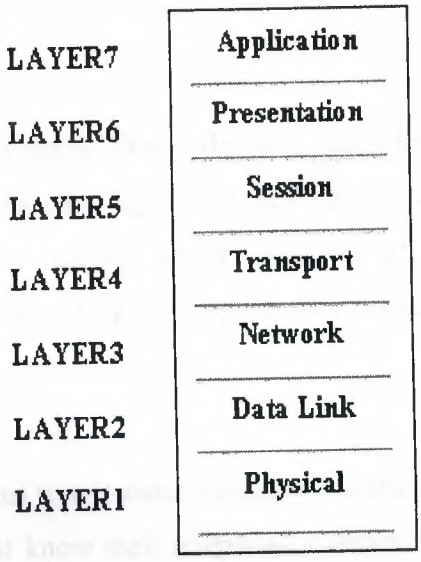

Figure 4.1 the ISO/OSI Reference Model

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 use 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 exploits 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 that the facility of host 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 mailbox 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 Dmain 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 is 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

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

NEAR EAST UNIVERSITY

Faculty of Engineering

Department of Computer Engineering

CRYPTOGRAPHY & SECURITY OVER NETWORK

Graduation Project
Com 400

Prepared By: Mohammad Maslat (20010690).

Supervisor: Assoc. Prof. Dr. Rahib Abyev.

Nicosia - 2006

Dedicated to my parents

LIST OF ABBREVIATIONS

MDC:	Modification Detection Codes.
MAC:	Message Authentication Codes.
ECB:	Electronic Codebook.
CBC:	Cipher Block Chaining.
DES:	Data Encryption Standard.
LFSR:	Linear Feedback Shift Register.
SPEKE:	Simple Password Exponential key Exchange.
DH-EKE:	Diffie-Hellman Encrypted Key Exchange.
DSA:	Digital Signature Methods.
FFT:	Fast Fourier Transform.
PGP:	Pretty Good Privacy.
RSA:	Rivest, Shamir, Adleman.
KE:	Keyless Encryption.
OTP:	One-Time Pads.
RSAP:	RSA Problem.
ISO:	International Standards Organization.
OSI:	Open Systems Interconnect.
TCP/IP:	Transport Control Protocol/Internet Protocol.
UDP:	User Datagram Protocol.
IETF:	Internet Engineering Task Force.

LIST OF ABBREVIATIONS

DNS:	Domain Name System.
ISP:	Internet Service Provider.
HTTP:	Hypertext Transfers Protocol.
SMTP:	Simple Mails Transfer Protocol.
DoS:	Denial-of-Service.
DMZ:	Demilitarized Zone.
FTP:	File Transfer Protocol.
ACL:	Access Control lists.
NAT:	Network Address Translation.
PAT:	Port Address Translation.
VPN:	Virtual Private Networks.

ACKNOWLEDGEMENT

First of all I would like to thanks Allah {God} for guiding me through my studies And who has given me the power and the patience to finish my bachelor degree's studies successfully.

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

Also, I feel proud to pay my regards to my project adviser "Assoc. Prof. Dr. Rahib Abyev ". He never disappointed me in any affair. He delivered me too much information and did his best of efforts to make me able to complete my project. Not to forget to give my thanks to the ***NEAR EAST UNIVERSITY*** *education staff especially to the computer engineering doctors for their helping to take this degree and to achieve this level of education.*

I will never forget the days that I have been in Cyprus, from the University to the good friends that I have enjoyed my 4 years with them. I would like to thank them for there kindness in helping me to complete my project:

My close friends; Eng.Baha Khalaf, Eng. Ala Mansour, Eng.Sa'ed Maslat and Also: Eng.Talal Khader, Khaled Abu Zagleh (Abu Sharks), AbdulLAtif Al Shamali, Omar Enbawi, Qusi, Mazen al Shawabkeh, Fadi Hamad, Tamer Maiah.

Also my special thanks to my all friends in Jordan.

ACKNOWLEDGEMENT

At the end I would like to thank my family again starting with my great Father: {ASA'D MASLAT}, the best mother in the world: {IBTISAM MASLAT}, As well as thanks to my brother Moayyad Maslat For his encourage, my sisters and not to forget their husbands, and best relatives (MOHAMMAD AL SHADFAN & NEDAL AL RABAY'A), I would like to give them my regards and appreciated their excellency and efforts with asking about me.

Mohammad Maslat.

3/2/2006

ABSTRACT

Cryptography algorithms are applied to protect a message or file from being read by network hackers, eavesdroppers. The encryption programs encrypt the text and will change the letters into symbols and other weird characters, so when someone opens the file they cannot read it. The interconnection of networks is an increasing trend in government and private industry. There is the obvious danger that connections made in such an extended network may increase the risk of a security compromise, with the owners unaware of the risk.

Network connections should therefore be protected, at a level based on the risk. The assumption must be that the connecting parties are to a certain degree hostile and have to be strictly constrained to the access for which the connection was agreed.

Although cryptography is fascinating and glamorous, because of its association with such things as espionage, diplomacy, and the higher levels of the military, it has a limited but important role in the area of network security.

The RSA cryptosystem, named after its inventors R. Rivest, A. Shamir, and L. Adleman, is the most widely used public-key cryptosystem. It may be used to provide both secrecy and digital signatures and its security is based on the intractability of the integer factorization. RSA encryption is most commonly used for the transport of symmetric-key encryption algorithm keys and for the encryption of small data items. The RSA cryptosystem has been patented in the U.S. and Canada. Several standards organizations have written, or are in the process of writing, standards that address the use of the RSA cryptosystem for encryption, digital signatures, and key establishment. For discussion of patent and standards issues related to RSA. The description of RSA algorithm is given in the thesis.

TABLE OF CONTENTS

Introduction

The origin of the word cryptology lies in ancient Greek. The word cryptology is made up of two components: "kryptos", which means hidden and "logos" which means word. Cryptology is as old as writing itself, and has been used for thousands of years to safeguard military and diplomatic communications. For example, the famous Roman emperor Julius Caesar used a cipher to protect the messages to his troops. Within the field of cryptology one can see two separate divisions: cryptography and cryptanalysis. The cryptographer seeks methods to ensure the safety and security of conversations while the cryptanalyst tries to undo the farmer's work by breaking his systems.

The main goals of modern cryptography can be seen as: user authentication, data authentication (data integrity and data origin authentication), non-repudiation of origin, and data confidentiality. In the following section we will elaborate more on these services. Subsequently we will explain how these services can be realized using cryptographic primitives.

A cryptographic system (or a cipher system) is a method of hiding data so that only certain people can view it. Cryptography is the practice of creating and using cryptographic systems. Cryptanalysis is the science of analyzing and reverse engineering cryptographic systems. The original data is called plaintext. The protected data is called cipher text. Encryption is a procedure to convert plaintext into cipher text. Decryption is a procedure to convert cipher text into plaintext. A cryptographic system typically consists of algorithms, keys, and key management facilities. There are two basic types of cryptographic systems: symmetric ("private key") and asymmetric ("public key").

Symmetric key systems require both the sender and the recipient to have the same key. This key is used by the sender to encrypt the data, and again by the recipient to decrypt the data. Key exchange is clearly a problem. How do you securely send a key that will enable you to send other data securely? If a private key is intercepted or stolen, the adversary can act

as either party and view all data and communications. You can think of the symmetric crypto system as akin to the Chubb type of door locks. You must be in possession of a key to both open and lock the door. Asymmetric cryptographic systems are considered much more flexible. Each user has both a public key and a private key.

Messages are encrypted with one key and can be decrypted only by the other key. The public key can be published widely while the private key is kept secret. If Alice wishes to send Bob a secret, she finds and verifies Bob's public key, encrypts her message with it, and mails it off to Bob. When Bob gets the message, he uses his private key to decrypt it. Verification of public keys is an important step. Failure to verify that the public key really does belong to Bob leaves open the possibility that Alice is using a key whose associated private key is in the hands of an enemy. Public Key Infrastructures or PKI's deal with this problem by providing certification authorities that sign keys by a supposedly trusted party and make them available for download or verification. Asymmetric ciphers are much slower than their symmetric counterparts and key sizes are generally much larger. You can think of a public key system as akin to a Yale type door lock. Anyone can push the door locked, but you must be in possession of the correct key to open the door.

The project is devoted the description of cryptographic algorithms, particularly RSA algorithm over network. The Goal of RSA Algorithm is to implement a demonstrable application that will perform the encryption and decryption of a text file using RSA Algorithm. I will give input as plaintext and it will generate the corresponding ciphertext. Ciphertext is decrypted to get the original plain text.

R.S.A. stands for Rivest, Shamir and Adleman - the three cryptographers who invented the first practical commercial public key cryptosystem. Today it is used in web browsers, email programs, mobile phones, virtual private networks, secure shells, and many other places. Exactly how much security it provides is debatable, but with sufficiently large keys you can be confident of foiling the vast majority of attackers. Until recently the use of RSA was very much restricted by patent and export laws. However, the patent has now expired and US export laws have been relaxed.

CHAPTER ONE

1. OVERVIEW OF CRYPTOGRAPHY SYSTEMS

1.1 Introduction

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 mentioned.

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 fsecurity 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 What Does Cryptography mean

Cryptography means 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.

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.

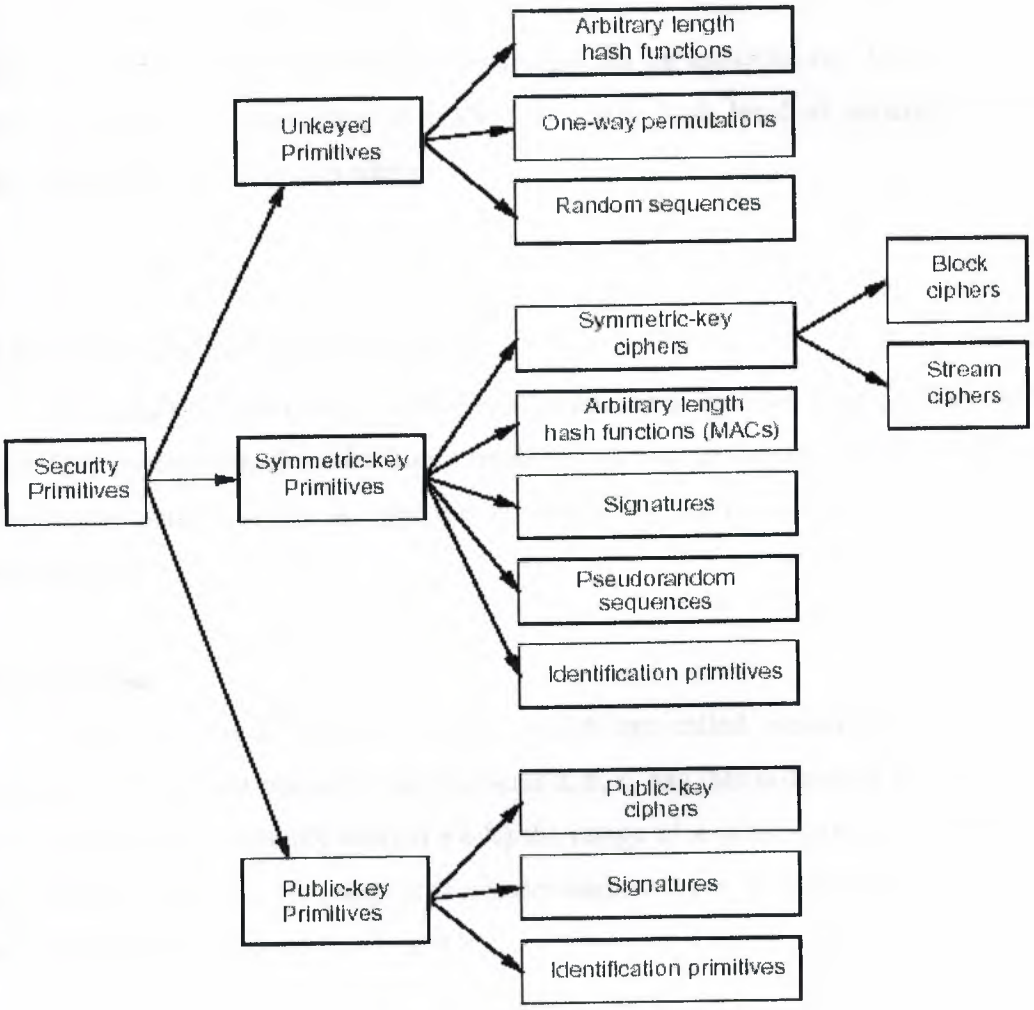

Figure 1.1 A taxonomy of cryptographic primitives.

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→Y.

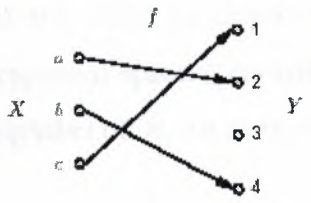

Figure 1.2 A function f from a set X to a set Y.

- 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→Y is 1-1 and Im (f) = Y, then f is called a bijection.

- One-way functions: A function f from a set X to a set Y is called a one-way function if f (x) is easy to compute for all $x \in X$ but for essentially all elements $y \in$ 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→S).
- Involutions: Involutions have the property that they are their own inverses. (i.e., f: S→ 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.
- C denotes a set called the cypertext space. C consists of strings of symbols from an alphabet; differ from the alphabet of $\mathcal{M}$. An element of C is called a cypertext.

1.3.2.2 Encryption and Decryption Transformations

- $\mathcal{K}$ 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 C, denoted by $\mathcal{E}e$.
- $\mathcal{D}_d$ denotes a bijection from 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 cypertext 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 ∈ M to Bob, she computes c = Ee (m) and transmits this to Bob. Upon receiving c, Bob computes D_d (c) = m and hence recovers the original message m.

The question arises as to why keys are necessary. If some particular encryption/decryption transformation is exposed then one does not have to redesign the entire scheme but simply change the key. Figure 1.3 provides a simple model of a two-party communication using encryption.

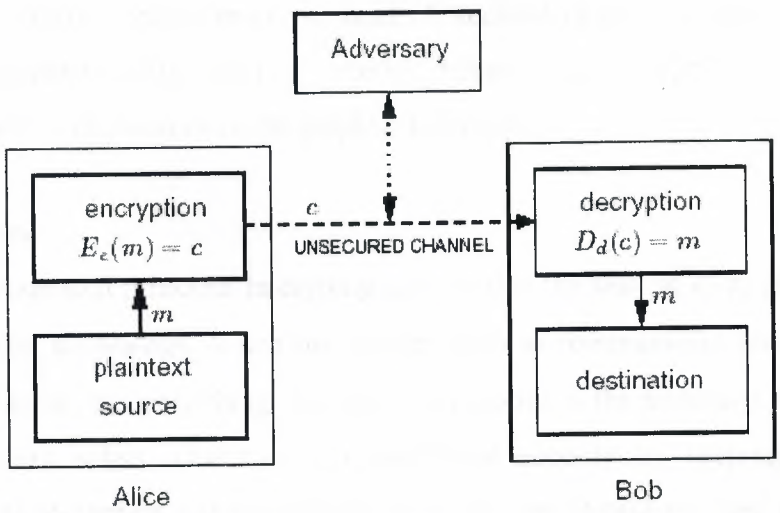

Figure 1.3 Schematic of a two-party communication.

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 $\mathcal{M}$; 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 cypertext within some appropriate time frame.

Trying all possible keys to see which one the communicating parties are using can break an encryption scheme. 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.

1.4 Symmetric-key Encryption

Consider an encryption scheme consisting of the sets of encryption and decryption transformations $\{\mathcal{E}e: e \in \mathcal{K}\}$ and $\{\mathcal{D}_d : d \in \mathcal{K}\}$, 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.

The block diagram of Figure 1.4, with the addition of the secure channel, can describe a two-party communication using symmetric-key encryption.

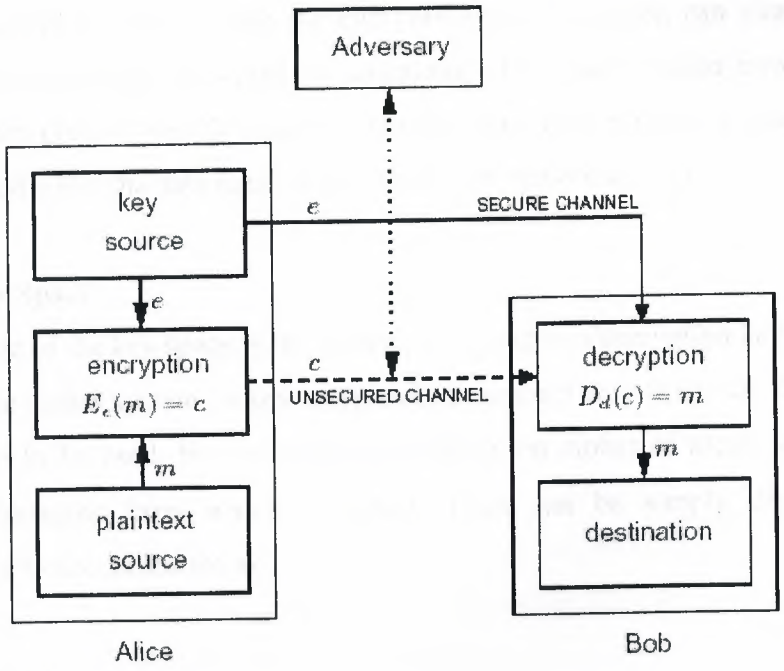

Figure 1.4 Two-party communication using encryption, with a secure channel

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 $S_{\mathcal{A}}$ and $V_{\mathcal{A}}$ provide a digital signature scheme for $\mathcal{A}$.

- $\mathcal{M}$ 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.
- $S_{\mathcal{A}}$ is a transformation from the message set $\mathcal{M}$ to the signature set $\mathcal{S}$, and is called a signing transformation for entity $\mathcal{A}$.

- $\mathcal{V}_A$ is a transformation from the set $\mathcal{M} \times S$ to the set {true, false} $\mathcal{V}_A$ 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 $\{\mathcal{E}e: e \in \mathcal{K}\}$ be a set of encryption transformations, and let $\{\mathcal{D}_d: d \in \mathcal{K}\}$ be the set of corresponding decryption transformations, where $\mathcal{K}$ is the key space. Consider any pair of associated encryption/decryption transformations $(\mathcal{E}e; \mathcal{D}d)$ and suppose that each pair has the property that knowing $\mathcal{E}e$ it is computationally infeasible, given a random ciphertext $c \in C$, to find the message $m \in \mathcal{M}$ such that $\mathcal{E}e(m) = c$. This property implies that given e it is infeasible to determine the corresponding decryption key d. $\mathcal{E}e$ 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

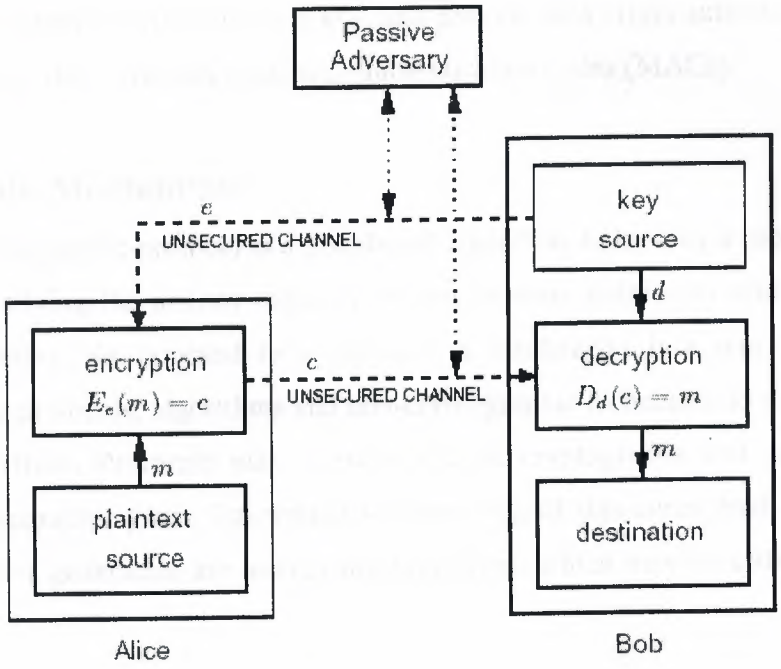

Figure 1.5 Encryption using public-key techniques.

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.

CHAPTER TWO

2. CRYPTOGRAPHY FUNCTIONS

2.1 Overview

In this chapter basic functions involved in cryptography are explained. Functions that 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.

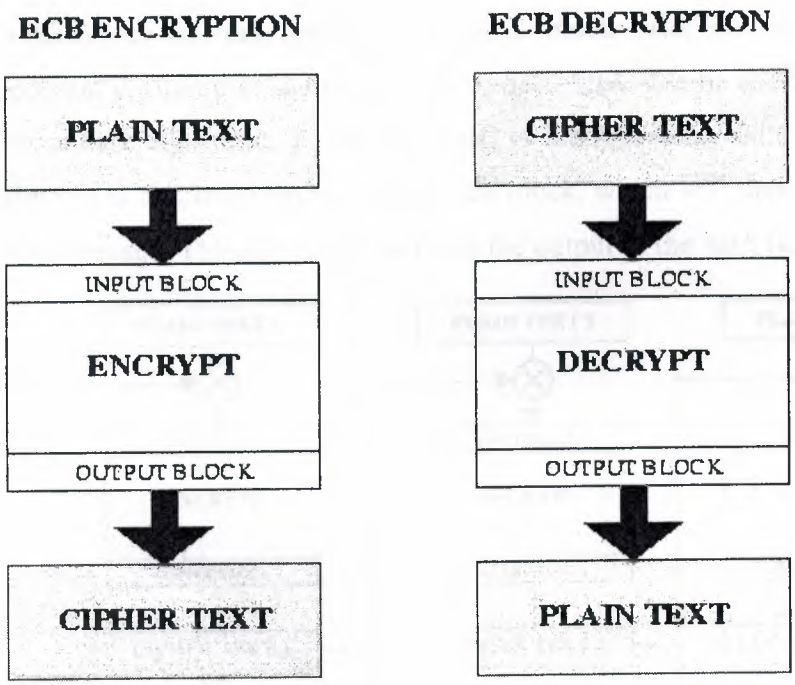

Figure 2.1: Shows a ECB Encryption/Decryption Model

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.

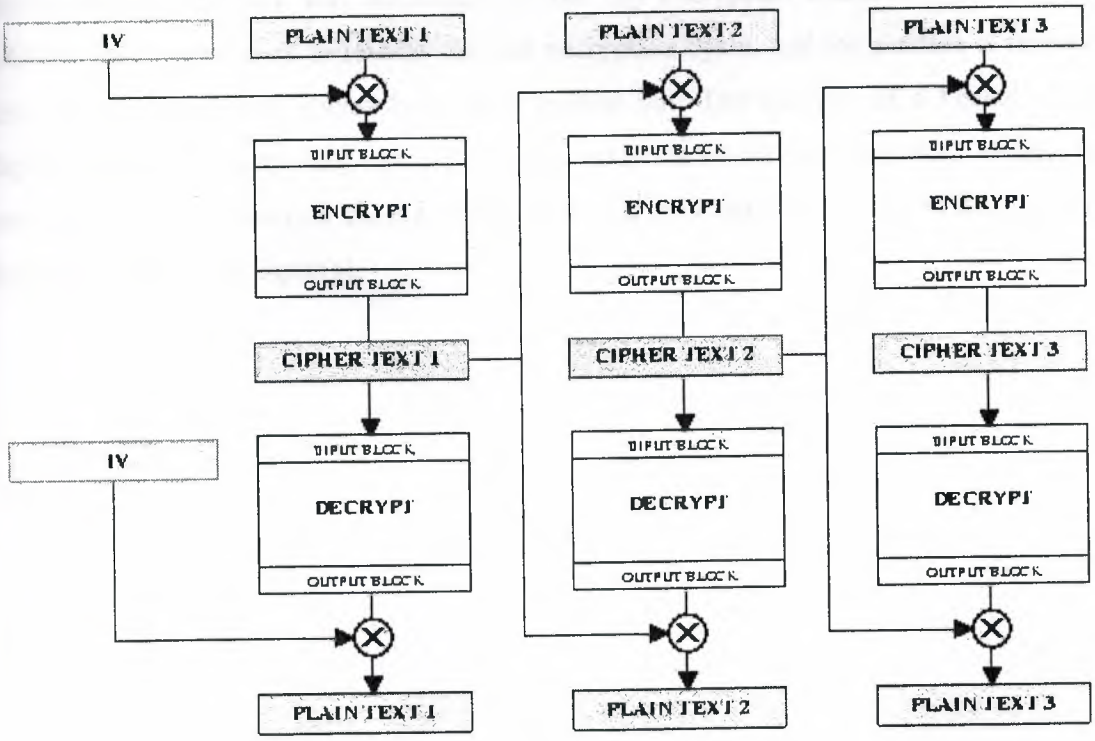

Figure 2.2: Shows a CBC Encryption/Decryption Model

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).

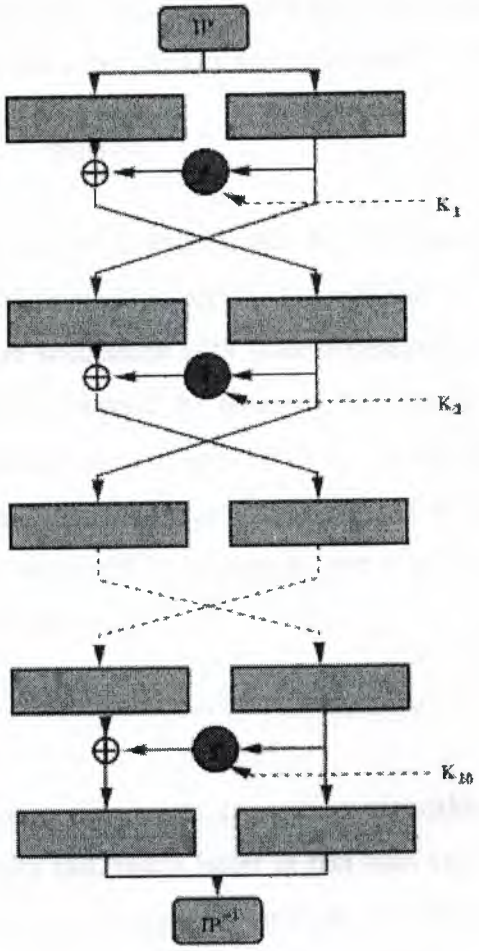

Figure 2.3: Shows a Feistel Model

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 combining the keystream with the plaintext, usually with the bitwise XOR operation, provides encryption. 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.

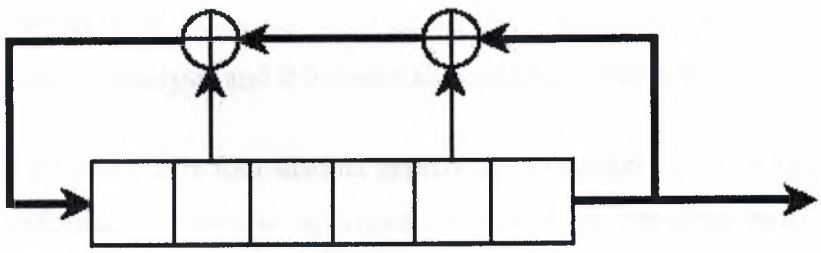

Figure 2.1: Shows a Linear Feed Back Register Model

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

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

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→B: h(h(K))
S6.	Bob verifies h(h(K)) is correct,	B→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:	$Q_A = g^{R}{}_A \bmod p$,	$A \rightarrow B$: $E_S (Q_A)$.
D2.	Bob computes:	$Q_B = g^{R}{}_B \bmod p$,	$B \rightarrow A$: $E_S (Q_B)$.
D3.	Alice computes:	$K = h(Q_B{}^{R}{}_A \bmod p)$	
D4.	Bob computes:	$K = h(Q_A{}^{R}{}_B \bmod p)$	

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 for 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 ($g^R{}_X$ mod 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:

$$K' = (E_{Si}^{-1}(E_S(g^{R}{}_B)))^{RA}$$

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:

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

$$= 2^{(S R_A)(X/S')} \bmod p$$

$$= 2^{(X R_A S/S')} \bmod p$$

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

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

Thus, when S' = S, K' = K. More generally, the attack works because the dictionary of passwords $\{S_1, S_2 \ldots S_n\}$ is equivalent to a dictionary of exponents $E = \{e_1, e_2 \ldots e_n\}$, 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 gc 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.

CHAPTER THREE

3. ENCRYPTION & DECRYPTION USING RSA ALGORITHM

3.1 Overview

RSA is a public-key cryptosystem developed by MIT professors: Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman in 1977 in an effort to help ensure Internet security. As Steve Burnett of RSA Data Security, Inc. described it, a cryptosystem is simply an algorithm that can convert input data into something unrecognizable (encryption), and convert the unrecognizable data back to its original form (decryption).

The RSA scheme is a block cipher in which the plaintext and cipher text are integers between 0 and n-1 for some n. A typical size for n is 1024 bits, or 309 decimal digits.

To encrypt data, enter the data ("plaintext") and an encryption key to the encryption portion of the algorithm. To decrypt the "cipher text," a proper decryption key is used at the decryption portion of the algorithm. Those keys, which contain simply a string of numbers, are called public key and private key, respectively. For example, suppose Alice intends to send e-mail to Bob. Through a public-key directory, she finds his public key. Then, she encrypts her message using the key and sends it to Bob. This public key, however, will not decrypt the cipher text. Knowledge of Bob's public key will not help an eavesdropper. In order for Bob to decrypt his cipher text, he must use his private key. If Bob wants to respond to Alice, he encrypts his message using her public key.

The challenge of public-key cryptography is developing a system in which it is impossible to determine the private key. This is accomplished through the use of a one-way function. With a one-way function, it is relatively easy to compute a result given some input values. In mathematical terms, given x, computing f(x) is easy, but given f(x), computing x is nearly impossible. The one-way function used in RSA is multiplication of prime numbers. It is easy to multiply two big prime numbers, but for most very large

primes, it is extremely time-consuming to factor them. Public-key cryptography uses this function by building a cryptosystem that uses two large primes to build the private key and the product of those primes to build the public key.

3.2 How does cryptographic algorithm work?

A *cryptographic algorithm,* or cipher, is a mathematical function used in the encryption and decryption process. A cryptographic algorithm works in combination with a *key* — a word, number, or phrase to encrypt the plaintext. The same plaintext encrypts to different cipher text with different keys. The security of encrypted data is entirely dependent on two things: the strength of the cryptographic algorithm and the secrecy of the key. A cryptographic algorithm, plus all possible keys and all the protocols that make it work comprise a *cryptosystem.* PGP is a cryptosystem.

3.3 Different types of Cryptosystems/Encryptions:

There are four ways of encryption that we mostly use in network system

- Pretty Good Privacy, PGP
- Rivest, Shamir, Adleman, RSA
- Keyless Encryption, KE
- One-Time Pads, OTP

3.3.1 PGP (Pretty Good Privacy)

Pretty Good Privacy. Up to this point we've talked about private-key cryptography (one key used y both parties). There was one problem with this kind of encryption: If the key was intercepted, a third party could decrypt the messages. So, the ideas of public-key cryptography were developed. Here's how it works...

Everyone has two keys: a public and a private key. When someone wants to send something to a recipient, they (the sender) encrypt it with the recipient's public key. Then the only way to decrypt it is with the recipient's private key. One of the other benefits to PGP is that it allows the sender to "sign" their messages. This proves that the message came from the sender and has not been altered in transport. Based on this theory, PGP

allows everyone to publicize their public keys, while keeping their private keys secret. The result is that anyone can encrypt a message to someone else, as long as they have that person's public key.

In actuality, PGP uses a series of private key, public key and one-way hash functions to encrypt a message. A one-way hash function takes some plaintext and translates it into a specific hash. The hash is unique to the message (like a fingerprint is to a person). The hash is also non-reversible, hence the name one-way. Let's run through an example of what PGP does to encrypt and decrypt an e-mail message. Our sender will be Chris and our receiver will be Brian.

- Chris writes his message.
- Chris uses a one-way hash function (such as MD5) to create a hash for the message.
- Chris, via RSA or some other digital signature algorithm, signs the hash with his private key.
- Chris merges the message and the signature, resulting in a new-signed message.
- A random encryption key is generated, the session key.
- Chris uses the session key to encrypt the message, using DES or some other private key method.
- Chris gets Brian's public key.
- Chris then encrypts the key with Brian's public key, via RSA or some other public key method.
- Chris merges the encrypted message and the encrypted key and mails it to Brian.

Once Brian receives the message he can have PGP decrypt it. Here's what it would do:

- Brian separates the encrypted message and the encrypted session key.
- Using RSA, Brian decrypts the session key.
- Using DES, Brian decrypts the message with the decrypted session key.
- Brian then separates the message and the signature.

- Using MD5, Brian calculates the hash value of the message.
- Brian gets Chris' public key.
- Via RSA, and Chris' public key, Brian decrypts the signature.
- Brian then compares the hash value and the decrypted signature. If they are the same, Brian knows that the message is authentic and has not been altered since Chris signed it.
-

3.3.2 RSA (Rivest, Shamir, Adleman)

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

$$C = M^e \bmod n$$

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

$$M = C^d \bmod n$$

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

values through equation (2) and get 19, our original plaintext! Try it yourself with other numbers.

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

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

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

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

3.4 The RSA Algorithm

Let's run the RSA algorithm through steps of how RSA algorithm procedure does work:

3.4.1 Key Generation

1.Generate two large prime numbers, p and q

2. Let n = pq

3. Let m = (p-1)(q-1)

4. Choose a small number e, coprime to m

5. Find d, such that de % m = 1

Publish e and n as the public key, Keep d and n as the secret k

Encryption

$C = P^e \% n$

Decryption

$P = C^d \% n$

x % y means the remainder of x divided by y

The reasons why this algorithm works are discussed in the mathematics section. Its security comes from the computational difficulty of factoring large numbers. To be secure, very large numbers must be used for p and q - 100 decimal digits at the very least.

I'll now go through a simple worked **example 1.**

a) Key Generation

1) Generate two large prime numbers, p and q

To make the example easy to follow I am going to use small numbers, but this is not secure. To find random primes, we start at a random number and go up ascending odd numbers until we find a prime. Lets have:

p = 7
q = 19

2) Let n = pq
n = 7 * 19
= 133

3) Let m = (p - 1)(q - 1)
m = (7 - 1)(19 - 1)
= 6 * 18
= 108

4) Choose a small number, e coprime to m

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

e = 2 => gcd(e, 108) = 2 (no)

e = 3 => gcd(e, 108) = 3 (no)

e = 4 => gcd(e, 108) = 4 (no)

e = 5 => gcd(e, 108) = 1 (yes!)

5) Find d, such that de % m = 1

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

n = 0 => d = 1 / 5 (no)

n = 1 => d = 109 / 5 (no)

n = 2 => d = 217 / 5 (no)

n = 3 => d = 325 / 5

= 65 (yes!)

To do this with big numbers, a more sophisticated algorithm called extended Euclid must be used.

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

b) Encryption

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

$C = P^e \% n$
$= 6^5 \% 133$
$= 7776 \% 133$
$= 62$

c) Decryption

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

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

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

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

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

3.5 From Applied Cryptography

$n = pq$

$ed \_ 1 \bmod (p \_ 1)(q \_ 1)$

$c = me \bmod n$

$m = cd \bmod n$

n is the product of p and q, two prime numbers.

The product of e and d divided by (p_1)(q _1) has a remainder of 1. C is the remainder of me divided by n; m is the remainder of cd divided by n.

3.5.1 The Product of Two Primes

n is the product of two prime numbers, p and q. n is made public as it is used in decryption as well as encryption. p and q should never be revealed.

Example:

p = 59, q = 67; n = 59 _ 67 = 3953

3.5.2 e and d, The Keys

e, the encryption key, is a random number relatively prime to (p _ 1)(q _ 1). d can be calculated to be de = 1 + x(p _ 1)(q _ 1), x is a throwaway value as it is \lost" in the modulus calculation. e is the public key and d is the private key.

Example, continued:

e = 7, (p _ 1)(q _ 1) = 3828

de = 1 + x(p _ 1)(q _ 1)

xd = 547, x = 1, d = 547

e = 7, d = 547

3.6 Key types

Two key types are employed in the primitives and schemes defined: *RSA public key* and *RSA private key*. Together, an RSA public key and an RSA private key form an *RSA key pair*.

This specification supports so-called "multi-prime" RSA where the modulus may have more than two prime factors. Better performance can be achieved on single processor platforms, but to a greater extent on multiprocessor platforms, where the modular exponentiations involved can be done in parallel.

3.6.1 RSA public key

RSA public key consists of two components:

n the RSA modulus, a positive integer

e the RSA public exponent, a positive integer

In a *valid RSA public key*, the RSA modulus n is a product of u distinct odd primes r_i, $i = 1, 2, \ldots, u$, where $u \geq 2$, and the RSA public exponent e is an integer between 3 and $n - 1$ satisfying GCD $(e, \lambda(n)) = 1$, where $\lambda(n) = \text{LCM}(r_1 - 1, \ldots, r_u - 1)$. By convention, the first two primes r_1 and r_2 may also be denoted p and q respectively.

3.6.2 RSA private key

An RSA private key may have either of two representations.

1. The first representation consists of the pair (n, d), where the components have the following meanings:

n the RSA modulus, a positive integer

d the RSA private exponent, a positive integer

2. The second representation consists of a quintuple $(p, q, dP, dQ, qInv)$ and a (possibly empty) sequence of triplets (r_i, d_i, t_i), $i = 3, \ldots, u$, one for each prime not in the quintuple, where the components have the following meanings:

p	the first factor, a positive integer
q	the second factor, a positive integer
dP	the first factor's CRT exponent, a positive integer
dQ	the second factor's CRT exponent, a positive integer
$qInv$	the (first) CRT coefficient, a positive integer
r_i	the i^{th} factor, a positive integer
d_i	the i^{th} factor's CRT exponent, a positive integer
t_i	the i^{th} factor's CRT coefficient, a positive integer

In a *valid RSA private key* with the first representation, the RSA modulus n is the same as in the corresponding RSA public key and is the product of u distinct odd primes

ri, i = 1, 2, …, u, where u ≥ 2. The RSA private exponent d is a positive integer less than n satisfying

$$e \cdot d \equiv 1 \pmod{\lambda(n)},$$

Where e is the corresponding RSA public exponent and $\lambda(n)$ is defined before.

In a valid RSA private key with the second representation, the two factors p and q are the *first two* prime factors of the RSA modulus n (i.e., r_1 and r_2), the CRT exponents dP and dQ are positive integers less than p and q respectively satisfying

$$e \cdot dP \equiv 1 \pmod{(p-1)}$$
$$e \cdot dQ \equiv 1 \pmod{(q-1)},$$

And the CRT coefficient $qInv$ is a positive integer less than p satisfying

$$q \cdot qInv \equiv 1 \pmod{p}.$$

If $u > 2$, the representation will include one or more triplets (r_i, d_i, t_i), $i = 3, \ldots, u$. The factors r_i are the additional prime factors of the RSA modulus n. Each CRT exponent d_i ($i = 3, \ldots, u$) satisfies

$$e \cdot d_i \equiv 1 \pmod{(r_i - 1)}.$$

Each CRT coefficient t_i ($i = 3, \ldots, u$) is a positive integer less than r_i satisfying

$$R_i \cdot t_i \equiv 1 \pmod{r_i},$$

Where $R_i = r_1 \cdot r_2 \cdot \ldots \cdot r_{i-1}$.

3.7 How fast is the RSA algorithm

An "RSA operation," whether encrypting, decrypting, signing, or verifying is essentially a modular exponentiation. This computation is performed by a series of modular multiplications.

In practical applications, it is common to choose a small public exponent for the public key. In fact, entire groups of users can use the same public exponent, each with a different modulus. (There are some restrictions on the prime factors of the modulus when the public exponent is fixed.) This makes encryption faster than decryption and verification faster than signing. With the typical modular exponentiation algorithms used to implement the RSA algorithm, public key operations take $O(k^2)$ steps, private key

operations take $O(k^3)$ steps, and key generation takes $O(k^4)$ steps, where k is the number of bits in the modulus. ``Fast multiplication'' techniques, such as methods based on the Fast Fourier Transform (FFT), require asymptotically fewer steps. In practice, however, they are not as common due to their greater software complexity and the fact that they may actually be slower for typical key sizes.

The speed and efficiency of the many commercially available software and hardware implementations of the RSA algorithm are increasing rapidly

By comparison, DES and other block ciphers are much faster than the RSA algorithm. DES is generally at least 100 times as fast in software and between 1,000 and 10,000 times as fast in hardware, depending on the implementation. Implementations of the RSA algorithm will probably narrow the gap a bit in coming years, due to high demand, but block ciphers will get faster as well.

3.8 Encryption and Decryption

The encryption and decryption math is more Straightforward, yet more demanding. The remainder of the message, or message block, value multiplied by itself key times divided by the modulus n is the encrypted or decrypted value, depending on which key is used. The value of the message, or message block, must be less than the value of n.

Example, continued:
Message: 0920 2000
09207 mod 3953 = 0307
20007 mod 3953 = 2497
0307547 mod 3953 = 0920
2497547 mod 3953 = 2000

3.8.1 The Mathematical Guts of RSA Encryption

Here's the algorithm behind RSA public key encryption:

Find *P* and *Q*, two large (1024-bit) prime numbers.
Choose *E* such that *E* is greater than 1, *E* is less than *PQ*, and *E* and *(P-1)(Q-1)* are *relatively prime*, which means they have no prime factors in common. *E* does not have to be prime, but it must be odd. *(P-1)(Q-1)* can't be prime because it's an even number.
Compute *D* such that *(DE - 1)* is evenly divisible by *(P-1)(Q-1)*. Mathematicians write this as $DE \equiv 1 \pmod{(P-1)(Q-1)}$, and they call *D* the *multiplicative inverse* of *E*. This is easy to do -- simply find an integer X which causes *D = (X(P-1)(Q-1) + 1)/E* to be an integer, then use that value of *D*.

The encryption function is *C = (T^E) mod PQ*, where *C* is the ciphertext (a positive integer), *T* is the plaintext (a positive integer), and ^ indicates exponentiation. The message being encrypted, *T* must be less than the modulus, *PQ*.

The decryption function is *T = (C^D) mod PQ*, where *C* is the ciphertext (a positive integer), *T* is the plaintext (a positive integer), and ^ indicates exponentiation.

The *public key* is the pair *(PQ, E)*. The *private key* is the number *D* (reveal it to no one). The product *PQ* is the *modulus* (often called N in the literature). *E* is the *public exponent*. *D* is the *secret exponent*.

The public key can be published freely, because there are no known easy methods of calculating *D*, *P*, or *Q* given only *(PQ, E)* (your public key). If *P* and *Q* were each 1024 bits long, it would be millions of years before the most powerful computers presently in existence can factor your modulus into *P* and *Q*.

3.8.2 RSA public-key encryption

The RSA cryptosystem, named after its inventors R. Rivest, A. Shamir, and L. Adleman, is the most widely used public-key cryptosystem. It may be used to provide both secrecy and digital signatures and its security is based on the intractability of the integer factorization.

3.8.2.1 Algorithm Key generations for RSA public-key encryption

Each entity creates an RSA public key and a corresponding private key.

Each entity A should do the following:

1. Generate two large random (and distinct) primes p and q, each roughly the same size.

2. Compute n = pq and _ = (p − 1)(q − 1).

3. Select a random integer e, 1 < e < _, such that gcd(e; _) = 1.

4. Use the extended Euclidean algorithm to compute the unique integer d, 1 < d < _, such that ed _ 1 (mod _).

5. A's public key is (n; e); A's private key is d.

Definition

The integers e and d in RSA key generation are called the *encryption exponent* and the *decryption exponent*, respectively, while n is called the *modulus*.

3.8.2.2 Algorithm RSA public-key encryption

B encrypts a message m for A, which A decrypts.

1. *Encryption.* B should do the following:

(a) Obtain A's authentic public key (n; e).

(b) Represent the message as an integer m in the interval [0; n − 1].

(c) Compute c = me mod n

(d) Send the cipher text c to A.

2. *Decryption.* To recover plaintext m from c, A should do the following:

(a) Use the private key d to recover m = cd mod n

Proof that decryption works. Since ed _ 1 (mod _), there exists an integer k such that ed = 1+k_. Now, if gcd(m; p) = 1 then by Fermat's theorem m^{p-1} _1 (modp):
Raising both sides of this congruence to the power k(q−1) and then multiplying both sides by m yields $m^{1+k(p-1)(q-1)}$ _ m (mod p):
On the other hand, if gcd(m; p) = p, then this last congruence is again valid since each side is congruent to 0 modulo p. Hence, in all cases m^{ed} _ m (mod p):

By the same argument, med _ m (mod q):

Finally, since p and q are distinct primes, it follows that

med _ m (mod n) and, hence,

cd _ (me)d _ m (mod n):

3.8.2.3 RSA encryption with artificially small parameter's example

Key generation. Entity A chooses the primes p = 2357, q = 2551, and computes n = pq = 6012707 and _ = (p−1)(q−1) = 6007800. A chooses e = 3674911 and, using the extended Euclidean algorithm, finds d = 422191 such that ed _ 1 (mod _). A's public key is the pair (n = 6012707; e = 3674911), while A's private key is d = 422191. *Encryption.* To encrypt a message m = 5234673, B uses an algorithm for modular exponentiation to compute:

c = me mod n = 52346733674911 mod 6012707 = 3650502; and sends this to A.

Decryption. To decrypt c, A computes

cd mod n = 3650502422191 mod 6012707 = 5234673.

3.8.2.4 Universal exponent

The number _ = lcm(p−1; q −1), sometimes called the *universal exponent* of n, may be used instead of _ = (p − 1)(q − 1) in RSA key generation Observe that _ is a proper divisor of _. Using _ can result in a smaller decryption exponent d, which may result in faster decryption However, if p and q are chosen at random, then gcd(p−1; q−1) is expected to be small, and consequently _ and _ will be roughly of the same size

3.9 Encryption Program

This program encrypted the plain text file of "text.txt" (Appendix) into the cyphertext file of "out.txt" (Appendix), which consisted of integer stream. At the same time, the public key pair and private key were generated and saved in the file of "key.txt" (Appendix). Finally, the decryption algorithm decrypted the cyphertext file of "out.txt"

into the file of "out_d.txt" (Appendix) with the aid of the private key pair and the private key.

In the main() function of "project_rsa.c", plaintext file was translated into ASCII code first as a stream of integers, and saved in the temporary file of "out". In order to hide the occurrence frequency of letters in English text, the letter was not encrypted respectively using RSA Encryption method. Instead, we extracted number with the fixed digits from the integer stream.

For example, using the key values in "project_rsa.c", the plaintext of "A3D4" can be translated as decimal integers stream of "65516852" according to the ASCII code. If we chose the 3 fixed digits, then this decimal integer stream can be separated as 655, 168, 52. It is noted that the last number should not be a 3 digits number. As such, we add '0' at the end of the number to expend as 520. Finally, the decimal integer stream can be separated as 655, 168, and 520. These three numbers were used for the encryption using RSA method. Accordingly, the cyphertext would be generated as: 28900 23450 20606 with the public key pair of (49447, 3513) and the private key of "11577".

Finally, the program read the cyphertext from the file of "out.txt". The original decimal integer stream can be decrypted using theses public and private keys, and save in the temporary file of "out_d". According to the ASCII code, all the English letters have ASCII code larger than 31 (We included the SPACE key) and less than 125. Then the plaintext can be easily translated from the original decimal integer stream. The decrypted plaintext was save in the file of "out_d.txt".

In order to access the program, the access process was inserted in the Encryption process in the main () function in "project_rsa.c".

Unlike other algorithm, the RSA test vector is so hard to find. To test this algorithm, and the implementation is in "rsa_test.c" (replace the main function of project_rsa.c), in which the key values are as following, public key = (3233, 17), private key = 2753. The plaintext value of 123 can be encrypted as 855.

3.9.1 Program List

Source Files:

project_rsa.c	main program
exponentiation.c	carry out the calculation of exponentiation
keys.c	generate the public and private keys
prime.c	find the primes use in the key generation
gcd.c	find the GCD of a and b
inverse.c	An implementation of the extended Euclidean algorithm
rsa_test.c	This is used for the test of the algorithm.

Header Files:

exponentiation.h

keys.h

gcd.h

inverse.h

3.10 RSA and related signature schemes

This section describes the RSA signature scheme and other closely related methods. The security of the schemes presented here relies to a large degree on the intractability of the integer factorization problem; the schemes presented include both digital signatures with message recovery and appendix.

3.10.1 The RSA signature scheme

The message space and ciphertext space for the RSA public-key encryption scheme are both Zn = f0; 1; 2; : : : ; n − 1g where n = pq is the product of two randomly chosen distinct prime numbers. Since the encryption transformation is a bijection, digital signatures can be created by reversing the roles of encryption and decryption. The RSA signature scheme is a deterministic digital signature scheme, which provides message

recovery .The signing spaceMS and signature space S, are both Zn. A redundancy function R: M−!Zn is chosen and is public knowledge.

3.10.1.1 Algorithm Key generation for the RSA signature scheme

Each entity creates an RSA public key and a corresponding private key.

Each entity A should do the following:

1. Generate two large distinct random primes p and q, each roughly the same size

2. Compute n = pq and _ = (p − 1)(q − 1).

3. Select a random integer e, 1 < e < _, such that gcd(e; _) = 1.

4. Use the extended Euclidean algorithm to compute the unique integer d, 1 < d < _, such that ed _ 1 (mod _).

5. A's public key is (n; e); A's private key is d.

3.10.1.2 Algorithm RSA signature generation and verification

Entity A signs a messagem2M. Any entity B can verify A's signature and recover the message m from the signature.

1. *Signature generation.* Entity A should do the following:

(a) Compute em = R(m), an integer in the range [0; n − 1].

(b) Compute s = emd mod n.

(c) A's signature for m is s.

2. *Verification.* To verify A's signature s and recover the message m, B should:

(a) Obtain A's authentic public key (n; e).

(b) Compute em = se mod n.

(c) Verify that em2MR; if not, reject the signature.

(d) Recover $m = R^{-1}(\bar{m})$.

Proof that signature verification works. If s is a signature for a message m, then $s \equiv \bar{m}^d \bmod n$ where $\bar{m} = R(m)$. Since $ed \equiv 1 \pmod{\phi}$, $s^e \equiv \bar{m}^{ed} \equiv \bar{m} \pmod{n}$. Finally, $R^{-1}(\bar{m}) = R^{-1}(R(m)) = m$.

3.10.2 Possible attacks on RSA signatures

(i) Integer factorization

If an adversary is able to factor the public modulus n of some entity A, then the adversary can compute _ and then, using the extended Euclidean algorithm, dedu the private key d from _ and the public exponent e by solving ed _ 1 (mod _). This constitutes a total break of the system. To guard against this, A must select p and q so that factoring n is a computationally infeasible task. For further information.

(ii) Multiplicative property of RSA

The RSA signature scheme (as well as the encryption method, cf has the following multiplicative property, sometimes referred to as the *homomorphic property*. If s1 = md1 mod n and s2 = md2 mod n are signatures on messages m1 and m2, respectively (or more properly on messages with redundancy added), then s = s1s2 mod n has the property that s = (m1m2)d mod n. Ifm = m1m2 has the proper redundancy (i.e.,m2MR), then s will be a valid signature for it.

Hence, it is important that the redundancy function R is not multiplicative, i.e., for essentially all pairs a; b 2 M, R(a _ b) 6= R(a)R(b). As

3.11 Security of RSA

This subsection discusses various security issues related to RSA encryption. Various attacks, which have been studied in the literature, are presented, as well as appropriate measures to counteract these threats.

(i) Relation to factoring

The task faced by a passive adversary is that of recovering plaintext from the corresponding ciphertext c, given the public information (n; e) of the intended receiver A. This is called the *RSA problem* (RSAP), which was introduced in x3.3. There is no efficient algorithm known for this problem.

One possible approach, which an adversary could employ to solving the RSA problem is to first factor n, and then compute e _ and d just as A did in Algorithm. Once d is obtained, the adversary can decrypt any ciphertext intended for A.

On the other hand, if an adversary could somehow compute d, then it could subsequently factor n efficiently as follows. First note that since ed _ 1 (mod _), there is an integer k such that ed − 1 = k_. Hence, by Fact 2.126(i), aed−1 _ 1 (mod n) for all a 2 Z_ n. Let ed − 1 = 2s t, where t is an odd integer. Then it can be shown that there exists an i 2 [1; s] such that a2i−1t 6_ _1 (mod n) and a2it _ 1 (mod n) for at least half of all a 2 Z_n; if a and i are such integers then gcd(a2i−1t − 1; n) is a non-trivial factor of n. Thus the adversary simply needs to repeatedly select random a 2 Z_ n and check if an i 2 [1; s] satisfying the above property exists; the expected number of trials before a non-trivial factor of n is obtained is 2. This discussion establishes the following.

Fact The problem of computing the RSA decryption exponent d from the public key (n; e), and the problem of factoring n, are computationally equivalent.

When generating RSA keys, it is imperative that the primes p and q be selected in such a way that factoring n = pq is computationally infeasible.

(ii) Small encryption exponent e.

In order to improve the efficiency of encryption, it is desirable to select a small encryption exponent e such as e = 3. A group of entities may all have the same encryption exponent e, however, each entity in the group must have its own distinct modulus (cf. x8.2.2(vi)). If an entity A wishes to send the same message m to three entities whose public moduli are n1, n2, n3, and whose encryption exponents are e = 3, then A would send ci = m3 mod ni, for i = 1; 2; 3. Since these moduli are most likely pairwise relatively

prime, an eavesdropper observing c1, c2, c3 can use Gauss's algorithm to find a solution x, 0 _ x < n1n2n3, to the three congruences 8<: x _ c1 (mod n1) x _ c2 (mod n2) x _ c3 (mod n3):

Since m3 < n1n2n3, by the Chinese remainder theorem it must be the case that x = m3. Hence, by computing the integer cube root of x, the eavesdropper can recover the plaintext m.

Thus a small encryption exponent such as e = 3 should not be used if the same message, or even the same message with known variations, is sent to many entities. Alternatively, to prevent against such an attack, a pseudorandomly generated bitstring of appropriate Length should be appended to the plaintext message prior to encryption; the pseudorandom bitstring should be independently generated for each encryption. This process is sometimes referred to as *salting* the message. Small encryption exponents are also a problem for small messages m, because ifm< n1=e, then m can be recovered from the ciphertext c = me mod n simply by computing the integer eth root of c; salting plaintext messages also circumvents this problem.

(iii) Forward search attack

If themessage space is small or predictable, an adversary can decrypt a ciphertext c by simply encrypting all possible plaintext messages until c is obtained. Salting the message as described above is one simple method of preventing such an attack.

(iv) Small decryption exponent d

As was the case with the encryption exponent e, it may seem desirable to select a small decryption exponent d in order to improve the efficiency of decryption.1 However, if gcd(p− 1; q − 1) is small, as is typically the case, and if d has up to approximately one-quarter as many bits as the modulus n, then there is an efficient algorithm for computing d from the public information (n; e). This algorithm cannot be extended to the case where d is approximately the same size as n. hence, to avoid this attack; the decryption exponent d should be roughly the same size as n.

(v) Multiplicative properties

Let m1 and m2 be two plaintext messages, and let c1 and c2 be their respective RSA encryptions. Observe that (m1m2)e _ me 1me 2 _ c1c2 (mod n):

In other words, the ciphertext corresponding to the plaintext m = m1m2 mod n is c = c1c2 mod n; this is sometimes referred to as the *homomorphic property* of RSA. This observation leads to the following *adaptive chosen-ciphertext attack* on RSA encryption.

Suppose that an active adversarywishes to decrypt a particular ciphertext c = me mod n intended for A. Suppose also that A will decrypt arbitrary ciphertext for the adversary, other than c itself. The adversary can conceal c by selecting a random integer x 2 Z_ n and computing c = cxe mod n. upon presentation of c, A will compute for the adversary

m = (c)d mod n. Since

m _ (c)d _ cd(xe)d _ mx (mod n);

The adversary can then compute m = mx−1 mod n.

This adaptive chosen-ciphertext attack should be circumvented in practice by imposing some structural constraints on plaintext messages. If a ciphertext c is decrypted to amessage not possessing this structure, then c is rejected by the decryptor as being fraudulent. Now, if a plaintext message m has this (carefully chosen) structure, then with high probability mx mod n will not for x 2 Z_ n. Thus the adaptive chosen-ciphertext attack described in the previous paragraph will fail because A will not decrypt c for the adversary. Provides a powerful technique for guarding against adaptive chosen-ciphertext and other kinds of attacks.

(vi) Common modulus attack

The following discussion demonstrates why it is imperative for each entity to choose its own RSA modulus n.

It is sometimes suggested that a central trusted authority should select a single RSA modulus n, and then distribute a distinct encryption/decryption exponent pair (ei; di) to each entity in a network. However, as shown in (i) above, knowledge of any (ei; di) pair allows for the factorization of the modulus n, and hence any entity could subsequently determine the decryption exponents of all other entities in the network. Also, if a

singlemessage were encrypted and sent to two or more entities in the network, then there is a technique by which an eavesdropper (any entity not in the network) could recover the message with high probability using only publicly available information.

(vii) Cycling attacks

Let c = me mod n be a ciphertext. Let k be a positive integer such that cek _ c (mod n); since encryption is a permutation on the message space f0; 1; : : : ; n − 1g such an integer

k must exist. For the same reason it must be the case that cek−1 _ m (mod n). This observation leads to the following *cycling attack* on RSA encryption. An adversary computes ce mod n, ce2 mod n, ce3 mod n; : : : until c is obtained for the first time. If cek mod n = c, then the previous number in the cycle, namely cek−1 mod n, is equal to the plaintext m.

A *generalized cycling attack* is to find the smallest positive integer u such that f = gcd(ceu − c; n) > 1. If ceu _ c (mod p) and ceu 6_ c (mod q) then f = p. Similarly, if ceu 6_ c (mod p) and ceu _ c (mod q) (8.2) then f = q. In either case, n has been factored, and the adversary can recover d and then m. On the other hand, if both ceu _ c (mod p) and ceu _ c (mod q); then f = n and ceu _ c (mod n). In fact, u must be the smallest positive integer k for which cek _ c (mod n). In this case, the basic cycling attack has succeeded and so m = ceu−1 mod n can be computed efficiently. Since is expected to occur much less frequently than the generalized cycling attack usually terminates before the cycling attack does. For this reason, the generalized cycling attack can be viewed as being essentially an algorithm for factoring n.

Since factoring n is assumed to be intractable, these cycling attacks do not pose a threat to the security of RSA encryption.

(viii) Message concealing

A plaintext message m, 0 _ m _ n−1, in the RSA public-key encryption scheme is said to be *unconcealed* if it encrypts to itself; that is, me _ m (mod n). There are always

some messages, which are unconcealed (for example m = 0, m = 1, and m = n − 1). In fact, the number of unconcealed messages is exactly

$[1 + \gcd(e - 1; p - 1)] \_ [1 + \gcd(e - 1; q - 1)]$:

Since e−1, p−1 and q −1 are all even, the number of unconcealed messages is always at least 9. If p and q are random primes, and if e is chosen at random (or if e is chosen to be a small number such as e = 3 or e = 216 + 1 = 65537), then the proportion of messages which are unconcealed by RSA encryption will, in general, be negligibly small, and hence unconcealed messages do not pose a threat to the security of RSA encryption in practice.

3.12 RSA encryption in practice

There are numerous ways of speeding up RSA encryption and decryption in software and hardware implementations. Some of these techniques are covered, including fast modular multiplication (x14.3), fast modular exponentiation (x14.6), and the use of the Chinese remainder theorem for faster decryption, Even with these improvements, RSA encryption/decryption is substantially slower than the commonly used symmetric-key encryption algorithms such as DES

In practice, RSA encryption is most commonly used for the transport of symmetric-key encryption algorithm keys and for the encryption of small data items.

The RSA cryptosystem has been patented in the U.S. and Canada. Several standards organizations have written, or are in the process of writing, standards that address the use of the RSA cryptosystemfor encryption, digital signatures, and key establishment. For discussion of patent and standards issues related to RSA,

3.12.1 Recommended size of modulus

Given the latest progress in algorithms for factoring integers (x3.2), a 512-bit modulus n provides onlymarginal security from concerted attack.

As of 1996, in order to foil the powerful quadratic sieve (x3.2.6) and number field sieve

(x3.2.7) Factoring algorithms, a modulus n of at least 768 bits is recommended. For long-term security, 1024-bit or larger moduli should be used.

3.12.2 Selecting primes

(i) As mentioned in x8.2.2(i), the primes p and q should be selected so that factoring n = pq is computationally infeasible. The major restriction on p and q in order to avoid the elliptic curve factoring algorithm (x3.2.4) is that p and q should be about the same bitlength, and sufficiently large. For example, if a 1024-bit modulus n is to be used, then each of p and q should be about 512 bits in length.

(ii) Another restriction on the primes p and q is that the difference p − q should not be too small. If p − q is small, then p _ q and hence p _ p n. Thus, n could be factored efficiently simply by trial division by all odd integers close to p n. If p and q are chosen at random, then p − q will be appropriately large with overwhelming probability.

(iii) In addition to these restrictions, many authors have recommended that p and q be strong primes. A prime p is said to be a *strong prime* if the following three conditions are satisfied:

(a) p − 1 has a large prime factor, denoted r;

(b) p + 1 has a large prime factor; and

(c) r − 1 has a large prime factor.

An algorithm for generating strong primes is presented in x4.4.2. The reason for condition

(a) is to foil Pollard's p−1 factoring algorithm(x3.2.3) which is efficient only if n has a prime factor p such that p − 1 is smooth. Condition (b) foils the p + 1 factoring algorithm mentioned on page 125 in x3.12, which is efficient only if n has a prime factor p such that p + 1 is smooth. Finally, condition (c) ensures that the cycling attacks described in x8.2.2(vii) will fail. If the prime p is randomly chosen and is sufficiently

arge, then both p−1 and p+1 can be expected to have large prime factors. In any case, while strong primes protect against the p−1 and p+1 factoring algorithms, they do not protect against their generalization, the elliptic curve-factoring algorithm (x3.2.4). The latter is successful in factoring n if a randomly chosen number of the same size as p (more precisely, this number is the order of a randomly selected elliptic curve defined over Zp) has only small prime factors. Additionally, it has been shown that the chances of a cycling attack succeeding are negligible if p and q are randomly chosen (cf. x8.2.2 (vii)). Thus, strong primes offer little protection beyond that offered by random primes. Given the current state of knowledge of factoring algorithms, there is no compelling reason for requiring the use of strong primes in RSA key generation. On the other hand, they are no less secure than random primes, and require only minimal additional running time to compute; thus there is little real additional cost in using them.

3.12.3 Small encryption exponents

(i) If the encryption exponent e is chosen at random, then RSA encryption using the repeated square-and-multiply algorithm takes k modular squaring and an expected k=2 (less with optimizations) modular multiplications, where k is the bit length of the modulus n. Encryption can be sped up by selecting e to be small and/or by selecting e with a small number of 1's in its binary representation.

(ii) The encryption exponent e = 3 is commonly used in practice; in this case, it is necessary that neither p−1 nor q−1 be divisible by 3. This results in a very fast encryption operation since encryption only requires 1 modular multiplication and 1 modular squaring. Another encryption exponent used in practice is e = 216 + 1 = 65537.

This number has only two 1's in its binary representation, and so encryption using the repeated square-and-multiply algorithm requires only 16 modular squaring and 1 modular multiplication. The encryption exponent e = 216 + 1 has the advantage over e = 3 in that it resists the kind of attack discussed in x8.2.2(ii), since it is unlikely the same message will be sent to 216+1 recipients.

CHAPTER FOUR

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 cannot work without the services provided by the layer below it.

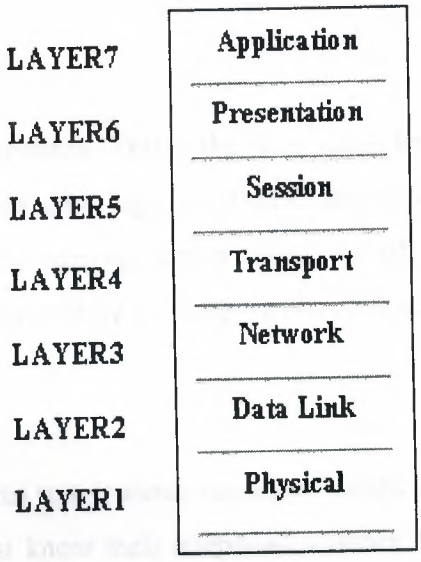

Figure 4.1 the ISO/OSI Reference Model

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 use 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 exploits 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 that the facility of host 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 mailbox 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 Dmain 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 is 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

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

.6 Types and Sources of Network Threats

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

.6.1 Denial-of-Service

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

4.7 Security Concepts and Technology

This section includes a brief description of network security concepts and technology. This information can be used to understand some of the security methods that are deployed throughout the network.

A comprehensive security approach requires that the company's different levels of management collectively create an enterprise-wide security approach by determining the appropriate security policies. The business side of the company typically uses policies to manage, so this concept is not unfamiliar to managers. A security policy defines the assets that need to be protected, who can have access to those assets, when they can have access, and how they are allowed to use the assets.

More important is the fact that the managers who own certain corporate assets (for example, customer or company data) have excellent insight into what policies should be designed to control access to these assets. Therefore, input from these managers is invaluable for securing the assets of the company. Unfortunately, there are many companies today where the security management of the company is totally in the hands of IT staff. While they are very knowledgeable and competent, the IT staff must have

input from the owners of the data to truly provide a comprehensive and consistent security management strategy. Without this input, the security of the company's assets may be at risk.

Once the high-level security policies have been determined, the security strategy can be developed from them. The security strategy should include a security plan that defines the tools and technologies to be used, and how they should be deployed. In addition, more specific access policies can be developed.

The security plan should include strategies that secure the perimeter of the enterprise, as well as strategies to secure the internal network. While the perimeter defense is a necessary piece of a complete security approach, the security strategy should not end there. Once intruders have access to the internal network, there must be security measures to prevent them from causing irreparable damage. A combination of security tools and technologies must be deployed throughout the network to ensure a secure network.

4.7.1 Firewalls

The concept of the firewall is much like the walled cities of medieval times, where an external perimeter was constructed to keep intruders out and to protect the residents within. The gates are designed both to control the entry of outsiders and to allow residents to leave the walled city. In addition, the gates provide limited-access points that are more easily defended against intruders.

Originally, many companies viewed firewalls as solid walls that would totally block outside entry to the enterprise. However, with the increased popularity of the Internet and the interactions of e-business, that approach is no longer acceptable. Administrators must now strike a balance between allowing required services through the firewall, while ensuring the security of the company's assets. As a result, the role of the firewall has evolved from being a solid perimeter wall to becoming the gates in the enterprise's perimeter wall.

A number of terms specific to firewalls and networking are going to be used oughout this section, so let's introduce them all together.

.1.1 Bastion Host

A general-purpose computer used to control access between the internal (private) twork (intranet) and the Internet (or any other untrusted network). Typically, these are sts running a flavor of the Unix operating system that has been customized in order to duce its functionality to only what is necessary in order to support its functions. Many the general-purpose features have been turned off, and in many cases, completely moved, in order to improve the security of the machine.

.7.1.2 Access Control List (ACL).

Many routers now have the ability to selectively perform their duties, based on a umber of facts about a packet that comes to it. This includes things like origination ddress, destination address, destination service port, and so on. These can be employed o limit the sorts of packets that are allowed to come in and go out of a given network.

.7.1.3 Demilitarized Zone (DMZ)

The DMZ is a critical part of a firewall: it is a network that is neither part of the ntrusted network, nor part of the trusted network. But, this is a network that connects the ntrusted to the trusted. The importance of a DMZ is tremendous: someone who breaks nto your network from the Internet should have to get through several layers in order to successfully do so. Those layers are provided by various components within the DMZ.

4.7.1.4 Proxy

This is the process of having one host act in behalf of another. A host that has the ability to fetch documents from the Internet might be configured as a *proxy server*, and host on the intranet might be configured to be *proxy clients*. In this situation, when a host

on the intranet wishes to fetch the web page, for example, the browser will make a connection to the proxy server, and request the given URL. The proxy server will fetch the document, and return the result to the client. In this way, all hosts on the intranet are able to access resources on the Internet without having the ability to direct talk to the Internet.

Now that the concept of firewalls has been described, it would be useful to have a basic understanding of how they work. The traffic coming into or going out of the corporate network originates from a location that is identified with an IP address (a unique network address). In addition, the traffic is composed of services that may be required by the enterprise, such as e-mail, File Transfer Protocol (FTP), Telnet, and many others. When setting up a firewall, the security administrator must define what services are to be allowed (both inbound and outbound), and whether to filter incoming and outgoing traffic based on IP addresses. The techniques that most firewalls use to filter incoming and outgoing traffic to the corporate networks are IP filtering, a proxy, or a combination of both methods.

4.7.1.5 IP Filtering

Every device on a TCP/IP network (the Internet, for example) is identified by a unique IP address. IP filtering is an access-control mechanism that filters network traffic based on IP addresses and requested services. It does this by using access control lists (ACLs), of which there are two types:

Host-based access control lists, which describe the services that are allowed or denied for each host or network. Service-based access lists, which describe the hosts or networks that are allowed or denied to use each service.

The firewall will reject any services or hosts that are denied access in the ACLs. Likewise, it will accept services from hosts that are allowed access in the ACLs. Network devices, such as firewalls and routers, can use ACLs to control access. In a recent Enterprise Management Associates study on security, 50% of the 100 respondents polled

ported that they use IP filtering. Of those respondents that use IP filtering, 86% of them se IP filtering on their firewalls.

ACL is almost like a guest list at an exclusive and high-security event. The list ontains the names of those "guests" who have been invited and are allowed to attend the vent. In addition, the guest list may also list services, such as the caterer, florist, or ntertainers, who should be allowed to enter. The guest list may even name specific eople who were not invited, and request that the security staff be especially vigilant to revent them from entering. It may also include instructions that certain services, such as he media, should not be allowed to enter. So the ACL acts like a guest list by naming who can and cannot have access, in addition to describing services that can and cannot have access through the firewall or router.

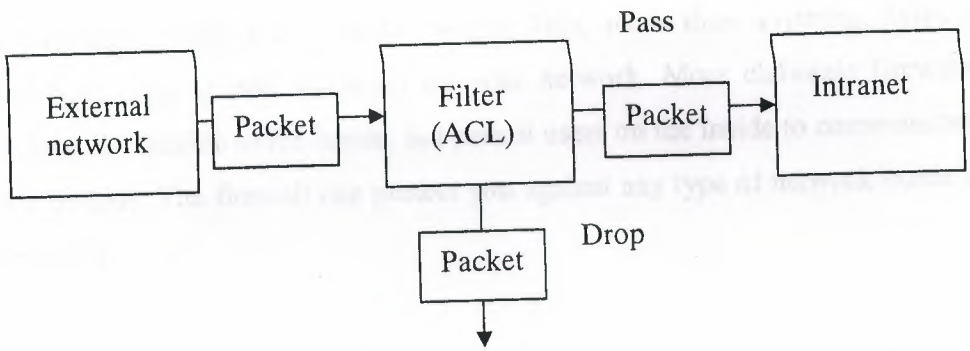

Figure 4.2 IP Filtering

To be effective, access control lists must be carefully and comprehensively constructed to ensure that unauthorized access and services are not allowed into the network. The ordering of the rules in the ACL is important because the first match that the firewall finds is executed. Creating and maintaining comprehensive ACLs can be a tedious task for security administrators of large and complex networks, especially if the definitions of ACLs are done manually. Because manually managing ACLs throughout the enterprise is difficult, in some cases only bare minimum ACLs are used, or they are not as widely deployed as they should be. To take full advantage of the benefits that IP

filtering can offer, security administrations need to use ACL management tools that facilitate easy deployment and administration of ACLs.

IP filtering provides flexibility, allowing administrators to create both simple access rules and a sophisticated set of rules to define what traffic will be allowed to pass through the firewall. In addition, IP filtering is a relatively fast method for controlling access because it is typically processed in the system kernel.

4.7.2 What Can A Firewall Protect Against?

Some firewalls permit only email traffic through them, thereby protecting the network against any attacks other than attacks against the email service. Other firewalls provide less strict protections, and block services that are known to be problems.

Generally, firewalls are configured to protect against unauthenticated Interactive logins from the "outside" world. This, more than anything, helps prevent vandals from logging into machines on your network. More elaborate firewalls block traffic from the outside to the inside, but permit users on the inside to communicate freely with the outside. The firewall can protect you against any type of network-borne attack if you unplug it.

Firewalls are also important since they can provide a single ``choke point'' where security and audit can be imposed. Unlike in a situation where a computer system is being attacked by someone dialing in with a modem, the firewall can act as an effective ``phone tap'' and tracing tool. Firewalls provide an important logging and auditing function; often they provide summaries to the administrator about what kinds and amount of traffic passed through it, how many attempts there were to break into it, etc.

This is an important point: providing this "choke point" can serve the same purpose on your network as a guarded gate can for your site's physical premises. That

means anytime you have a change in "zones" or levels of sensitivity, such a checkpoint is appropriate. A company rarely has only an outside gate and any receptionist or security staff to check badges on the way in. If there are layers of security on your site, it's reasonable to expect layers of security on your network.

4.7.3 What Can't A Firewall Protect Against?

Firewalls can't protect against attacks that don't go through the firewall. Many corporations that connect to the Internet are very concerned about proprietary data leaking out of the company through that route. Unfortunately for those concerned, a magnetic tape can just as effectively be used to export data. Many organizations that are terrified (at a management level) of Internet connections have no coherent policy about how dial-in access via modems should be protected. It's silly to build a 6-foot thick steel door when you live in a wooden house, but there are a lot of organizations out there buying expensive firewalls and neglecting the numerous other back-doors into their network. For a firewall to work, it must be a part of a consistent overall organizational security architecture. Firewall policies must be realistic and reflect the level of security in the entire network. For example, a site with top secret or classified data doesn't need a firewall at all: they shouldn't be hooking up to the Internet in the first place, or the systems with the really secret data should be isolated from the rest of the corporate network.

Another thing a firewall can't really protect you against is traitors or idiots inside your network. While an industrial spy might export information through your firewall, he's just as likely to export it through a telephone, FAX machine, or floppy disk. Floppy disks are a far more likely means for information to leak from your organization than a firewall! Firewalls also cannot protect you against stupidity. Users who reveal sensitive information over the telephone are good targets for social engineering; an attacker may be able to break into your network by completely bypassing your firewall, if he can find a "helpful" employee inside who can be fooled into giving access to a modem pool. Before deciding this isn't a problem in your organization, ask yourself how much trouble a

contractor has getting logged into the network or how much difficulty a user who forgot his password has getting it reset. If the people on the help desk believe that every call is internal, you have a problem.

Lastly, firewalls can't protect against tunneling over most application protocols to poorly written clients. There are no magic bullets and a firewall is not an excuse to not implement software controls on internal networks or ignores host security on servers. Tunneling ``bad'' things over HTTP, SMTP, and other protocols is quite simple and trivially demonstrated. Security isn't "fire and forget".

4.7.4 Application Level Firewall

An application level firewall evaluates network packets for valid data at the application layer before allowing a connection. The firewall examines the data in all network packets at the application layer and maintains complete connection state and sequencing information. Other security items such as user password and service requests that appear in the application layer data can be validated by the firewall. Specialized application software and proxy services are included in most application layer firewalls. Proxy services manage traffic through a firewall for a specific service such as HTTP or FTP. Proxy services can provide increased access control, detailed checks for valid data, and generate audit records about the traffic they transfer because the proxy services are specific to the protocol that they are designed to forward.

An application level firewall analyzes the complete command set for a single protocol in application space. When an incoming network packet is received it moves up the hardened network stack until it reaches the highest protocol layer found in the packet. After the network stack finishes processing the packet, its data is passed from kernel space to application space then to the proxy server that is listening on a specific TCP or UDP port. Next the proxy service processes the data it has received. The data is compared to the acceptable command set rules, as well as to host and user permission rules. The proxy determines whether to accept or deny the packet based on the results of

the rules comparison. Based on how it was configured, the proxy may also perform other functions such as URL filtering, data modification, authentication logging, and HTTP object caching. A proxy service consists of the proxy server, proxy client and protocol analysis modes of operation.

A proxy server and a proxy client are two components that are typically implemented as a single executable for each application proxy. A proxy server acts as the end server for all connection requests originated on a trusted network by a real client. Rather than allowing users to communicate directly with the other servers on the Internet, all communication between the internal users on the trusted network and the Internet passes through the proxy server. When the internal user wants to connect to an external service such as FTP or Telnet they send a request to the proxy server for the connection. The proxy server decides whether to permit or deny the request based on an evaluation of a set of rules that is managed for the individual network service. Proxy servers only allow those packets through that comply with the protocol definitions because the servers understand the protocol of the service they are evaluating. The proxy client is the component that talks to the server on the external network on behalf of the real client on the trusted network. The proxy server evaluates a real client's request for a service against the policy rules defined for that proxy and determines whether to approve the request. The proxy server forwards the request to the proxy client if the request is approved. The proxy client contacts the real server on the external network on behalf of the client. The proxy client relays requests from the proxy server to the real server and relays responses from the real server to the proxy server. Then the proxy server relays the requests and responses between the proxy client and the real client.

Proxy services never allow direct connection between the real client on the trusted network and the real server on the external network. Proxy services force all network packets to be examined and filtered for suitability. All communication between the real user and the real service are handled by the proxy service. The proxy service is

transparent to the user on the trusted network and the real service on the external network.

Proxy services are implemented on the top of the firewall host's network stack and operate only in the application space of the operating system. Proxy services are slower than packet filtering because each packet in a session is subjected to an examination process. Each network packet must pass through the low-level protocols in the kernel before being passed up the stack to application space. Once in the application space the proxies perform a thorough inspection of the packet headers and packet data. After inspection and acceptance the packet must travel back down to the kernel, and then back down the stack for distribution. Additional checks can be performed by application level firewalls to ensure that a network packet has not been spoofed. Application level firewalls can often perform network address translation.

Application level firewall technology using proxy services has several advantages. Proxy services enforce high level protocols such as HTTP and FTP. Information about the communications passing through the firewall server is maintained by the proxy service. Proxy services can permit access to certain network services, while denying access to others. Packet data can be processed and manipulated by proxy services. Internal IP addresses are shielded from the external world because proxy services do not allow direct communications between external server and internal computers. Administrators are able to monitor attempts to violate the firewall's security policies using the audit records that proxy services can generate.

Although application level firewalls provide increased security over a packet filtering firewall there are some disadvantages to using an application level firewall. Application level firewalls are slower since inbound data is processed by the application and by its proxy. A new proxy usually must be written for each protocol that is to pass through the firewall. This can cause the number of available network services and their scalability to be limited. Proxy services are vulnerable to operating system and application level bugs.

Most application level firewalls require extensive support from the operating system to run correctly. The firewalls need support from TCP/IP, Win32, Winsock, NDIS, and the standard C library. The security of the firewall server can be effected by problems in these operating system components.

4.7.4.1 Proxy Servers

What are proxy servers? Proxy servers are basically a buffer between you, the user, and the Internet resources that you are accessing. Proxies are the software agents that act on the behalf of a user. A typical proxy will accept a connection from a user, make a decision as to whether or not the user or client IP address is permitted to use the proxy, and then it completes a connection on behalf of the user to a remote destination. The data that is requested by the user first filters through the proxy and then it is delivered to the client.

One of the main reasons for using a proxy server is to give access to the Internet from within a firewall. A proxy server operating at the application-level makes a firewall permeable for users in the organization. They allow the users inside an organization to have access to information outside of the firewall without creating a possible security breach for the company. A proxy will ensure that there is no security hole where some attacker could get into the subnet. Each user inside a company will have access to the proxy. Access to the proxy can be customized to best fit each user's security and computing needs, giving different degrees of access to each user. Proxying is a standard method for getting through Firewalls, rather than having each client gets customized to support a special firewall product or method.

Proxy servers can be used for many other reasons. One of those reasons is that you can permit or restrict each user's access based on their IP address. Proxies contain a table, which dictates the security level allowed by each user. A sample security table is shown below as shown in Table 4.1.

Client IP Address	Security Level
3.3.3.55	Full Intranet, Full Internet
3.3.3.56	3.3.3.XX Subnet Only
3.3.3.57	Full Intranet, No Internet
3.3.3.58	Full Intranet, Full Internet
3.3.3.59	3.3.3.XX Subnet Only

Table 4.1 Security Table of IP Addresses

This demonstrates how only selected clients have full access to the Internet or Intranet, while other users access can be limited to a specific subnet of the Intranet.

Another reason to use proxy servers is for the increased performance of the network. Proxies can cache documents that are frequently accessed by clients inside the firewall. Document caching is storing the most often requested file in memory where multiple users can access the document, rather than having to go get the document every time it is requested. This will cut down on the amount of disk space used, which will allow faster travel through the network. Caching will be discussed in further detail later in the report.

Proxies are placed between a physical connection point to the Internet and the connection point to the local network. What this means is that all communication from the network to the Internet or other Intranets is routed through the proxy. The actual workstations do not have a direct physical connection to the Internet, and therefore it is not possible for them to communicate in a direct way. The proxy server delivers the requests from the local net to the Internet as if it were the original requester. The proxy is a special HTTP server that typically runs on a firewall machine. It waits for requests from inside the firewall, and then sends them to the remote server, gets the response and sends it back to the client. This process is illustrated in the Figure 4.3 below.

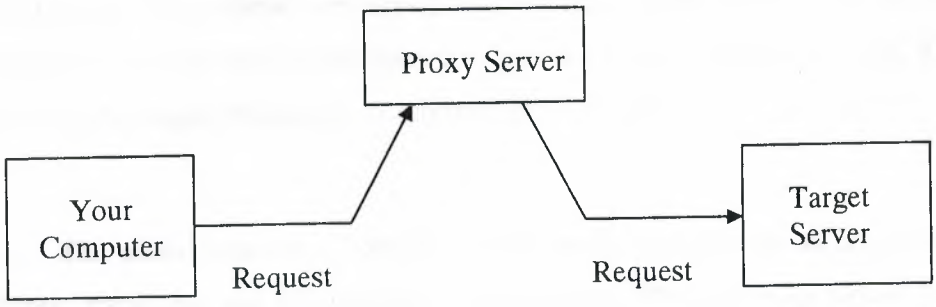

Figure 4.3 Proxy Server making a request

Proxy servers protect the network against hackers and those out to steal information from a company's internal network by disguising the client's IP address. The proxy does this by creating an address table where the client's true IP address is paired with a false IP address that is assigned to the proxy server. The philosophy behind the false IP address is that if an intruder were to capture the IP address during transmission of the message they would not actually gain access into the company's network. By using the fake address the intruder would only be able to access the proxy server. Below in table 4.2, is an example of an address table stored in the proxy server's memory.

Client IP Address	Proxy Assigned IP Address
3.3.3.55	192.168.21.21
3.3.3.57	192.168.21.22
3.3.3.47	192.168.21.23
Not Assigned	192.168.21.24
Not Assigned	192.168.21.25

Table 4.2 Address Table

The use of this table protects the IP addresses that are internal to the network from being "stolen" by users outside of the network. If by chance someone outside of the network were to gain access to these IP addresses the internal network would still be safe

cause none of the clients are physically connected to the Internet. The IP addresses the tacker would have would take them to some alternate site on the Internet. This is one of e main advantages to routing all Internet traffic through the proxy server.

The actual process a typical request would go through is comprised of many fferent steps that are all transparent to the user. The first step would be the actual quest of an Internet site by the client. This client would travel through the subnet to the roxy server assigned to that subnet. Once the proxy receives the request it will take the ient's IP address and store it into the address table. After storing the client's IP address ie proxy will then substitute a fake IP address for the client's IP address and continue outing the request until it reaches its final destination. After the external sight processes ie request the response is routed back to the proxy server. Now the proxy server will erform a series of data integrity checks to ensure that the information received from utside the network should be allowed into the internal network. After validating the data searches the address table for the correct IP address and routes the information to the lient. The entire process is transparent to the user and does not significantly slow down ie processing and transmission speeds of the request.

.7.4.2 Circuit-level Gateways

One step above standard packet filtering firewalls, but still considered part of the ame architecture, are circuit level gateways, otherwise known as "stateful packet nspection" firewalls. In the circuit-level firewall, all connections are monitored and only hose connections that are found to be valid are allowed to pass through the firewall.

This generally means that a client behind the firewall can initiate any type of session, out clients outside the firewall cannot see or connect to a machine protected by the firewall. Stateful inspections usually occur at the Network Layer, thus making it fast and preventing suspect packets from traveling up the protocol stack. Unlike static packet filtering, however, stateful inspection makes its decisions based on all the data in the

cket (corresponding to all the levels of the OSI stack). Using this information, the ewall builds dynamic state tables.

It uses these tables to keep track of the connections that go through the firewall rather an allowing all packets that meet the rule set's requirements to pass, it allows only those ckets, which are part of a valid, established connection. Packet filtering firewalls are pular because they tend to be inexpensive, fast, and relatively easy to configure and aintain.

7.4.3 Application-Level Gateway

An application-level proxy server provides all the basic proxy features and also ovides extensive packet analysis. When packets from the outside arrive at the gateway, ey are examined and evaluated to determine if the security policy allows the packet to ter into the internal network. Not only does the server evaluate IP addresses, it also oks at the data in the packets to stop hackers from hiding information in the packets.

A typical application-level gateway can provide proxy services for applications d protocols like Telnet, FTP (file transfers), HTTP (Web services), and SMTP (e-mail). ote that a separate proxy must be installed for each application-level service (some endors achieve security by simply not providing proxies for some services, so be careful your evaluation). With proxies, security policies can be much more powerful and exible because all of the information in packets can be used by administrators to write e rules that determine how packets are handled by the gateway. It is easy to audit just bout everything that happens on the gateway. You can also strip computer names to hide ternal systems, and you can evaluate the contents of packets for *appropriateness* and ecurity.

7.4.4 Network Address Translation (NAT)

Firewalls using NAT and/or Port Address Translation (PAT) completely hide the etwork protected by the firewall by using many-to-one address translation. In most NAT mplementations there is a single public IP address used for the entire network. All

packets going outside the network have their internal IP addresses hidden for security, so any incoming packets are delivered to the network's public IP address. To handle ensuing port conflicts, PAT needs to be added to NAT.

NAT is the translation of an IP address used within one network to a different IP address known within another network. This is easier to understand by referring to the following diagram:

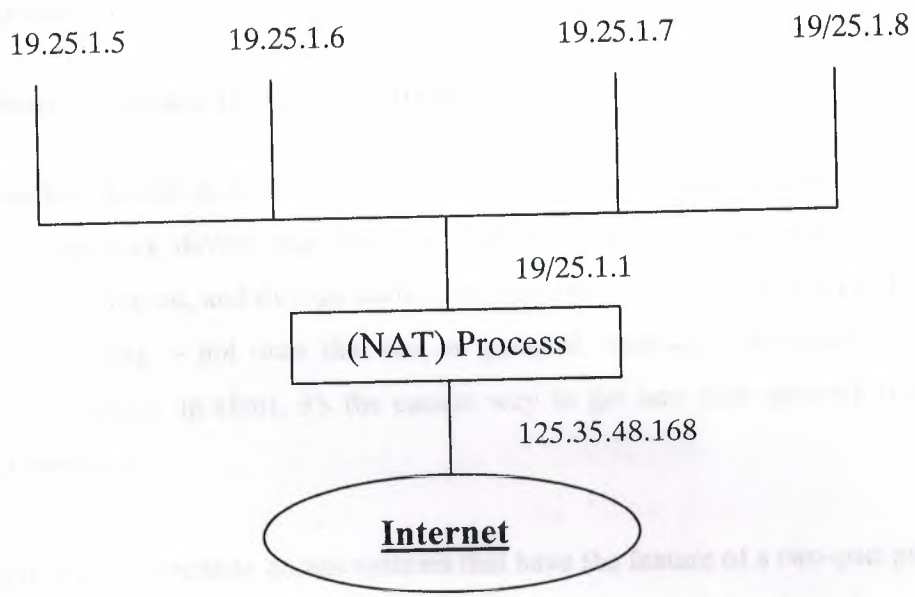

Figure 4.4 Network Address Translation

The NAT enabled router has an IP address of 10.25.1.1 for the inside network and an address of 125.35.48.168 for the outside network. Anytime a host on the inside network (10.25.1.0) makes a request to the outside network (the Internet in this instance) NAT will translate the 10.25.1.0 address to 125.35.48.168. From the inside looking out, the machines can access any host on the external network directly, while from the outside looking in it appears that all in and outbound traffic is originating from the single IP address on the router.

A disadvantage of NAT is that it can't properly pass protocols containing IP address information in the data portion of the packet.

4.8 Secure Network Devices

It's important to remember that the firewall only one entry point to your network. Modems, if you allow them to answer incoming calls, can provide an easy means for an attacker to sneak around (rather than through) your front door (or, firewall). Just as castles weren't built with moats only in the front, your network needs to be protected at all of its entry points.

4.8.1 Secure Modems; Dial-Back Systems

If modem access is to be provided, this should be guarded carefully. The terminal server, or network device that provides dial-up access to your network needs to be actively administered, and its logs need to be examined for strange behavior. Its password need to be strong -- not ones that can be guessed. Accounts that aren't actively used should be disabled. In short, it's the easiest way to get into your network from remote: guard it carefully.

There are some remote access systems that have the feature of a two-part procedure to establish a connection. The first part is the remote user dialing into the system, and providing the correct userid and password. The system will then drop the connection, and call the authenticated user back at a known telephone number. Once the remote user's system answers that call, the connection is established, and the user is on the network. This works well for folks working at home, but can be problematic for users wishing to dial in from hotel rooms and such when on business trips.

Other possibilities include one-time password schemes, where the user enters his userid, and is presented with a ``challenge,'' a string of between six and eight numbers.

He types this challenge into a small device that he carries with him that looks like a calculator. He then presses enter, and a ``response" is displayed on the LCD screen. The user types the response, and if all is correct, he login will proceed. These are useful devices for solving the problem of good passwords, without requiring dial-back access. However, these have their own problems, as they require the user to carry them, and they must be tracked, much like building and office keys.

No doubt many other schemes exist. Take a look at your options, and find out how what the vendors have to offer will help you enforce your security policy effectively.

4.8.2 Crypto-Capable Routers

A feature that is being built into some routers is the ability to session encryption between specified routers. Because traffic traveling across the Internet can be seen by people in the middle who have the resources (and time) to snoop around, these are advantageous for providing connectivity between two sites, such that there can be secure routes.

4.8.3 Virtual Private Networks

Given the ubiquity of the Internet, and the considerable expense in private leased lines, many organizations have been building VPNs (Virtual Private Networks). Traditionally, for an organization to provide connectivity between a main office and a satellite one, an expensive data line had to be leased in order to provide direct connectivity between the two offices. Now, a solution that is often more economical is to provide both offices connectivity to the Internet. Then, using the Internet as the medium, the two offices can communicate.

The danger in doing this, of course, is that there is no privacy on this channel, and it's difficult to provide the other office access to "internal" resources without providing those resources to everyone on the Internet.

VPNs provide the ability for two offices to communicate with each other in such a way that it looks like they're directly connected over a private leased line. The session between them, although going over the Internet, is private (because the link is encrypted), and the link is convenient, because each can see each other's internal resources without showing them off to the entire world.

A number of firewall vendors are including the ability to build VPNs in their offerings, either directly with their base product, or as an add-on. If you have needed to connect several offices together, this might very well be the best way to do it.

CONCLUSION

Many business executives see Internet services fueling the next major wave of e-business vth and process efficiencies. It groups expect Internet services to significantly reduce lication integration costs, and technology providers envision robust demand for a new eration of internet services-enabled offerings.

Despite all this forward motion, concerns about security are major barrier to adoption. er all, many enterprises have not fully mastered security measures in their existing e- siness environment .now, internet services technology makes it even easier to expose tical data and business process to the outside word, potentially increasing security risk.

There is strong momentum to bring Internet services technology into the mainstream of twork computing. Thus many companies tried to support authentication and authorization some things called cryptographic algorithms and digital signatures.

The project has included four chapters followed by a RSA algorithm developed by the thor to allow the server and client to authenticate each other and to negotiate an encryption gorithm and cryptographic keys before the application protocol transmit.

Chapter one has presented an overview of cryptography systems and the main definitions ave been described.

Chapter two has presented a cryptography functions and some of the mathematical unction for encryptions and decryption the data.

Chapter tree has described the encryption & decryption using RSA algorithms, and the main keys types for RSA algorithms in details.

Finally, Chapter four has described the network security and some general in formations and risks about the network and it's security details.

REFERENCES

1] Handbook of Applied Cryptography by A. Menezes, P. van Oorschot, and S. Vanstone n 1996.

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

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

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

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

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

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

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

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

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

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

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

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

4] J. Orlin Grabbe The DES Algorithm Illustrated
mepage: http://orlingrabbe.com. Retrieved October 25, 2003.

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

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