CRYPTOGRAPHY AND NETWORK SECURITY LECTURE NOTES

for

Bachelor of Technology

in

Computer Science and Engineering

&

Information Technology

Department of Computer Science and Engineering & Information Technology

Veer Surendra Sai University of Technology

(Formerly UCE, Burla)

Burla, Sambalpur, Odisha

Lecture Note Prepared by: **Prof. D. Chandrasekhar Rao**

Dr. Amiya Kumar Rath

Dr. M. R. Kabat

DISCLAIMER

This document does not claim any originality and cannot be used as a substitute for prescribed textbooks. The information presented here is merely a collection by the committee members for their respective teaching assignments. Various sources as mentioned at the end of the document as well as freely available material from internet were consulted for preparing this document. The ownership of the information lies with the respective authors or institutions.

CRYPTOGRAPHY AND NETWORK SECURITY

BCS- **(3-0-1) Credit-4**

Module I **(12 LECTURES)**

Introduction to the Concepts of Security: The need for security, Security Approaches, Principles of Security, Types of Attacks. Cryptographic Techniques: Plain Text and Cipher Text, Substitution Techniques, Transposition Techniques, Encryption and Decryption, Symmetric and Asymmetric Key Cryptography, Steganography, Key Range and Key Size, Possible Types of Attacks.

Module II **(8 LECTURES)**

Computer-based Symmetric Key Cryptographic Algorithms: Algorithm Types and Modes, An overview of Symmetric Key Cryptography, DES, International Data Encryption Algorithm (IDEA), RC5, Blowfish, AES, Differential and Linear Cryptanalysis.

Module III **(8 LECTURES)**

Computer-based Asymmetric Key Cryptography: Brief History of Asymmetric Key Cryptography, An overview of Asymmetric Key Cryptography, The RSA Algorithm, Symmetric and Asymmetric Key Cryptography Together, Digital Signatures, Knapsack Algorithm, Some other Algorithms.

Module IV **(12 LECTURES)**

Public Key Infrastructure: Digital Certificates, Private Key Management, The PKIX Model, Public Key Cryptography Standards, XML, PKI and Security. Internet Security Protocols: Basic Concepts, Secure Socket Layer, SHTTP, Time Stamping Protocol, Secure Electronic Transaction, SSL versus SET, 3-D Secure Protocol, Electronic Money, E-mail Security, Wireless Application Protocol (WAP) Security, Security in GSM.

Text Books:

1. Cryptography and Network Security – by Atul Kahate – TMH.
2. Data Communications and Networking- by Behourz A Forouzan

Reference Book:

1. Cyber Security Operations Handbook – by J.W. Rittiaghouse and William M.Hancok – Elseviers.

MODULE - I

INTRODUCTION

Computer data often travels from one computer to another, leaving the safety of its protected physical surroundings. Once the data is out of hand, people with bad intention could modify or forge your data, either for amusement or for their own benefit.

Cryptography can reformat and transform our data, making it safer on its trip between computers. The technology is based on the essentials of secret codes, augmented by modern mathematics that protects our data in powerful ways.

- **Computer Security** - generic name for the collection of tools designed to protect data and to thwart hackers
- **Network Security** - measures to protect data during their transmission
- **Internet Security** - measures to protect data during their transmission over a collection of interconnected networks

Security Attacks, Services and Mechanisms

To assess the security needs of an organization effectively, the manager responsible for security needs some systematic way of defining the requirements for security and characterization of approaches to satisfy those requirements. One approach is to consider three aspects of information security:

Security attack – Any action that compromises the security of information owned by an organization.

Security mechanism – A mechanism that is designed to detect, prevent or recover from a security attack.

Security service – A service that enhances the security of the data processing systems and the information transfers of an organization. The services are intended to counter security attacks and they make use of one or more security mechanisms to provide the service.

Basic Concepts

Cryptography The art or science encompassing the principles and methods of transforming an intelligible message into one that is unintelligible, and then retransforming that message back to its original form

Plaintext The original intelligible message

Cipher text The transformed message

Cipher An algorithm for transforming an intelligible message into one that is unintelligible by transposition and/or substitution methods

Key Some critical information used by the cipher, known only to the sender& receiver

Encipher (encode) The process of converting plaintext to cipher text using a cipher and a key

Decipher (decode) the process of converting cipher text back into plaintext using a cipher and a key

Cryptanalysis The study of principles and methods of transforming an unintelligible message back into an intelligible message *without* knowledge of the key. Also called **code breaking**

Cryptology Both cryptography and cryptanalysis

Code An algorithm for transforming an intelligible message into an unintelligible one using a code-book

Cryptography

Cryptographic systems are generally classified along 3 independent dimensions:

Type of operations used for transforming plain text to cipher text

All the encryption algorithms are based on two general principles: **substitution**, in which each element in the plaintext is mapped into another element, and **transposition**, in which elements in the plaintext are rearranged.

The number of keys used

If the sender and receiver uses same key then it is said to be **symmetric key (or) single key (or) conventional encryption**.

If the sender and receiver use different keys then it is said to be **public key encryption**.

The way in which the plain text is processed

A **block cipher** processes the input and block of elements at a time, producing output block for each input block.

A **stream cipher** processes the input elements continuously, producing output element one at a time, as it goes along.

Cryptanalysis

The process of attempting to discover X or K or both is known as cryptanalysis. The strategy used by the cryptanalysis depends on the nature of the encryption scheme and the information available to the cryptanalyst.

There are various types of cryptanalytic attacks based on the amount of information known to the cryptanalyst.

Cipher text only – A copy of cipher text alone is known to the cryptanalyst.

Known plaintext – The cryptanalyst has a copy of the cipher text and the corresponding plaintext.

Chosen plaintext – The cryptanalysts gains temporary access to the encryption machine. They cannot open it to find the key, however; they can encrypt a large number of suitably chosen plaintexts and try to use the resulting cipher texts to deduce the key.

Chosen cipher text – The cryptanalyst obtains temporary access to the decryption machine, uses it to decrypt several string of symbols, and tries to use the results to deduce the key.

STEGANOGRAPHY

A plaintext message may be hidden in any one of the two ways. The methods of steganography conceal the existence of the message, whereas the methods of cryptography render the message unintelligible to outsiders by various transformations of the text.

A simple form of steganography, but one that is time consuming to construct is one in which an arrangement of words or letters within an apparently innocuous text spells out the real message.

e.g., (i) the sequence of first letters of each word of the overall message spells out the real (Hidden) message.
(ii) Subset of the words of the overall message is used to convey the hidden message.

Various other techniques have been used historically, some of them are

Character marking – selected letters of printed or typewritten text are overwritten in pencil. The

marks are ordinarily not visible unless the paper is held to an angle to bright light.

Invisible ink – a number of substances can be used for writing but leave no visible trace until heat or some chemical is applied to the paper.

Pin punctures – small pin punctures on selected letters are ordinarily not visible unless the paper is held in front of the light. Typewritten correction ribbon – used between the lines typed with a black ribbon, the results of typing with the correction tape are visible only under a strong light.

Drawbacks of steganography

Requires a lot of overhead to hide a relatively few bits of information.

Once the system is discovered, it becomes virtually worthless.

SECURITY SERVICES

The classification of security services are as follows:

Confidentiality: Ensures that the information in a computer system and transmitted information are accessible only for reading by authorized parties.

E.g. Printing, displaying and other forms of disclosure.

Authentication: Ensures that the origin of a message or electronic document is correctly identified, with an assurance that the identity is not false.

Integrity: Ensures that only authorized parties are able to modify computer system assets and transmitted information. Modification includes writing, changing status, deleting, creating and delaying or replaying of transmitted messages.

Non repudiation: Requires that neither the sender nor the receiver of a message be able to deny the transmission.

Access control: Requires that access to information resources may be controlled by or the target system.

Availability: Requires that computer system assets be available to authorized parties when needed.

SECURITY MECHANISMS

One of the most specific security mechanisms in use is cryptographic techniques. Encryption or encryption-like transformations of information are the most common means of providing security. Some of the mechanisms are

1 Encipherment

2 **Digital Signature**

3 **Access Control**

SECURITY ATTACKS

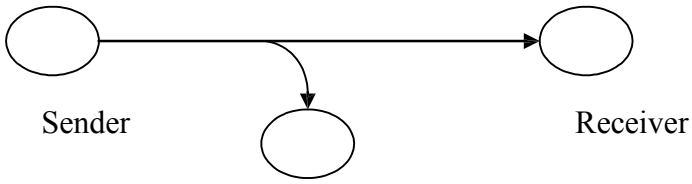
There are four general categories of attack which are listed below.

Interruption

An asset of the system is destroyed or becomes unavailable or unusable. This is an attack on availability e.g., destruction of piece of hardware, cutting of a communication line or Disabling of file management system.

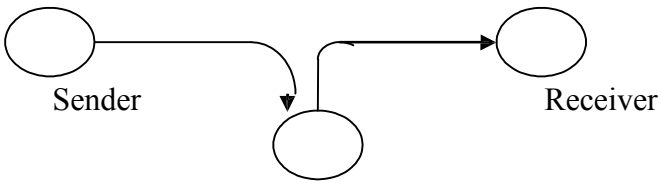
Interception

An unauthorized party gains access to an asset. This is an attack on confidentiality. Unauthorized party could be a person, a program or a computer.e.g., wire tapping to capture data in the network, illicit copying of files



Modification

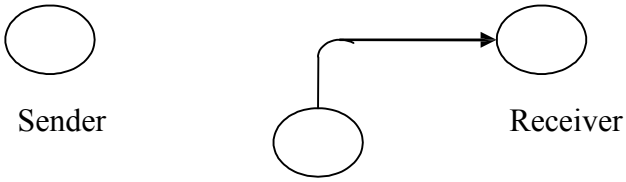
An unauthorized party not only gains access to but tampers with an asset. This is an attack on integrity. e.g., changing values in data file, altering a program, modifying the contents of messages being transmitted in a network.



Eavesdropper or forger

Fabrication

An unauthorized party inserts counterfeit objects into the system. This is an attack on authenticity. e.g., insertion of spurious message in a network or addition of records to a file.



Eavesdropper or forger

Cryptographic Attacks

Passive Attacks

Passive attacks are in the nature of eavesdropping on, or monitoring of, transmissions. The goal of the opponent is to obtain information that is being transmitted. Passive attacks are of two types:

Release of message contents: A telephone conversation, an e-mail message and a transferred file may contain sensitive or confidential information. We would like to prevent the opponent from learning the contents of these transmissions.

Traffic analysis: If we had encryption protection in place, an opponent might still be able to observe the pattern of the message. The opponent could determine the location and identity of communication hosts and could observe the frequency and length of messages being exchanged. This information might be useful in guessing the nature of communication that was taking place.

Passive attacks are very difficult to detect because they do not involve any alteration of data. However, it is feasible to prevent the success of these attacks.

Active attacks

These attacks involve some modification of the data stream or the creation of a false stream. These attacks can be classified in to four categories:

Masquerade – One entity pretends to be a different entity.

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

Modification of messages – Some portion of message is altered or the messages are delayed or recorded, to produce an unauthorized effect.

Denial of service – Prevents or inhibits the normal use or management of communication facilities. Another form of service denial is the disruption of an entire network, either by disabling the network or overloading it with messages so as to degrade performance.

It is quite difficult to prevent active attacks absolutely, because to do so would require physical protection of all communication facilities and paths at all times. Instead, the goal is to detect them and to recover from any disruption or delays caused by them.

Symmetric and public key algorithms

Encryption/Decryption methods fall into two categories.

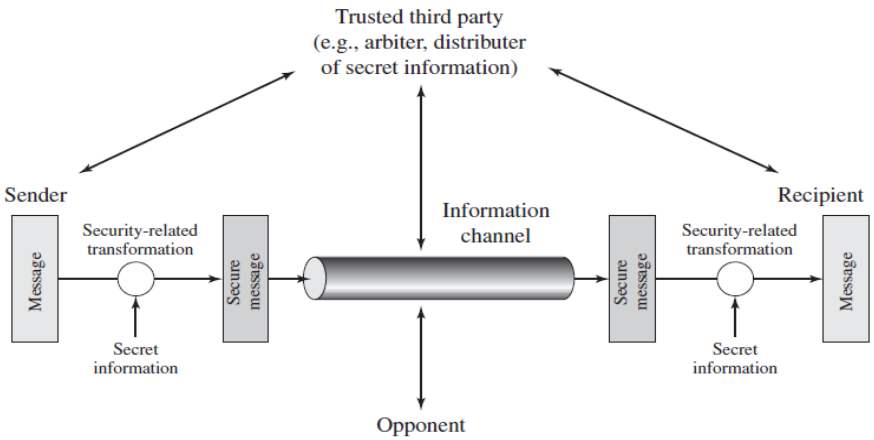
Symmetric key

Public key

In symmetric key algorithms, the encryption and decryption keys are known both to sender and receiver. The encryption key is shared and the decryption key is easily calculated from it. In many cases, the encryption and decryption keys are the same.

In public key cryptography, encryption key is made public, but it is computationally infeasible to find the decryption key without the information known to the receiver.

A MODEL FOR NETWORK SECURITY



A message is to be transferred from one party to another across some sort of internet. The two parties, who are the principals in this transaction, must cooperate for the exchange to take place. A logical information channel is established by defining a route through the internet from source to destination and by the cooperative use of communication protocols (e.g., TCP/IP) by the two principals.

Using this model requires us to:

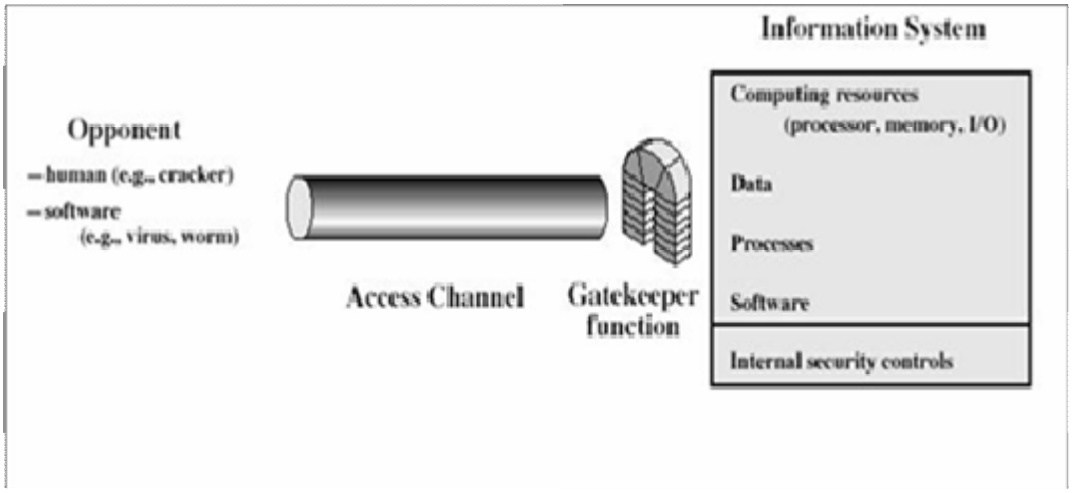
– design a suitable algorithm for the security transformation

– generate the secret information (keys) used by the algorithm

– develop methods to distribute and share the secret information

– specify a protocol enabling the principals to use the transformation and secret information for a security service

MODEL FOR NETWORK ACCESS SECURITY



Using this model requires us to:

– select appropriate gatekeeper functions to identify users

– implement security controls to ensure only authorized users access designated information or resources

- **Trusted computer systems can be used to implement this model**

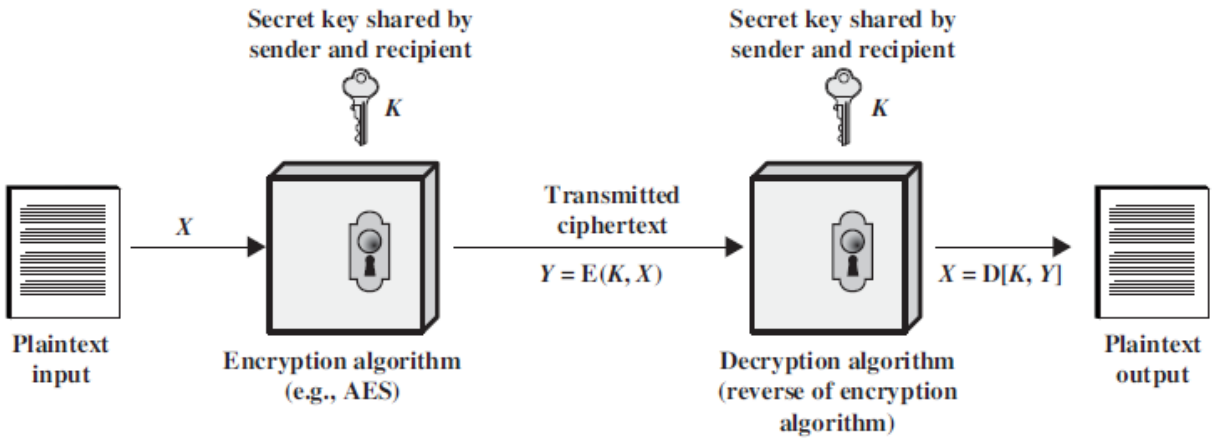
CONVENTIONAL ENCRYPTION

- Referred conventional / private-key / single-key
- Sender and recipient share a common key

All classical encryption algorithms are private-key was only type prior to invention of public-key in 1970"**plaintext** - the original message
Some basic terminologies used:

- **cipher text** - the coded message
- **Cipher** - algorithm for transforming plaintext to cipher text
- **Key** - info used in cipher known only to sender/receiver
- **encipher (encrypt)** - converting plaintext to cipher text
- **decipher (decrypt)** - recovering cipher text from plaintext
- **Cryptography** - study of encryption principles/methods

- **Cryptanalysis (code breaking)** - the study of principles/ methods of deciphering cipher text *without* knowing key
  - **Cryptology** - the field of both cryptography and cryptanalysis



Here the original message, referred to as plaintext, is converted into apparently random nonsense, referred to as cipher text. The encryption process consists of an algorithm and a key. The key is a value independent of the plaintext. Changing the key changes the output of the algorithm. Once the cipher text is produced, it may be transmitted. Upon reception, the cipher text can be transformed back to the original plaintext by using a decryption algorithm and the same key that was used for encryption. The security depends on several factors. First, the encryption algorithm must be powerful enough that it is impractical to decrypt a message on the basis of cipher text alone. Beyond that, the security depends on the secrecy of the key, not the secrecy of the algorithm.

- **Two requirements for secure use of symmetric encryption:**

– A strong encryption algorithm

– A secret key known only to sender / receiver

$Y = E_K(X)$

$X = D_K(Y)$

- **assume encryption algorithm is known**
- **implies a secure channel to distribute key**

A source produces a message in plaintext, X = [X1, X2… XM] where M are the number of letters in the message. A key of the form K = [K1, K2… KJ] is generated. If the key is generated at the source, then it must be provided to the destination by means of some secure channel.

With the message X and the encryption key K as input, the encryption algorithm forms the cipher text Y = [Y1, Y2, YN]. This can be expressed as

$Y = E_K(X)$

The intended receiver, in possession of the key, is able to invert the transformation:

$X = D_K(Y)$

An opponent, observing Y but not having access to K or X, may attempt to recover X or K or both. It is assumed that the opponent knows the encryption and decryption algorithms.

If the opponent is interested in only this particular message, then the focus of effort is to recover X by generating a plaintext estimate. Often if the opponent is interested in being able to read future messages as well, in which case an attempt is made to recover K by generating an estimate.

CLASSICAL ENCRYPTION TECHNIQUES

There are two basic building blocks of all encryption techniques: substitution and transposition.

SUBSTITUTION TECHNIQUES

A substitution technique is one in which the letters of plaintext are replaced by other letters or by numbers or symbols. If the plaintext is viewed as a sequence of bits, then substitution involves replacing plaintext bit patterns with cipher text bit patterns.

Caesar cipher (or) shift cipher

The earliest known use of a substitution cipher and the simplest was by Julius Caesar. The Caesar cipher involves replacing each letter of the alphabet with the letter standing 3 places further down the alphabet.

e.g., plain text : pay more money

Cipher text: SDB PRUH PRQHB

Note that the alphabet is wrapped around, so that letter following „z" is „a".

For each plaintext letter p, substitute the cipher text letter c such that

C = E(p) = (p+3) mod 26

A shift may be any amount, so that general Caesar algorithm is

C = E (p) = (p+k) mod 26

Where k takes on a value in the range 1 to 25. The decryption algorithm is simply

P = D(C) = (C-k) mod 26

Playfair cipher

The best known multiple letter encryption cipher is the playfair, which treats digrams in the plaintext as single units and translates these units into cipher text digrams. The playfair

algorithm is based on the use of 5x5 matrix of letters constructed using a keyword. Let the keyword be „monarchy". The matrix is constructed by filling in the letters of the keyword (minus duplicates) from left to right and from top to bottom, and then filling in the remainder of the matrix with the remaining letters in alphabetical order.

The letter „i" and „j" count as one letter. Plaintext is encrypted two letters at a time

According to the following rules:

Repeating plaintext letters that would fall in the same pair are separated with a

Filler letter such as „x".

Plaintext letters that fall in the same row of the matrix are each replaced by the letter to the right, with the first element of the row following the last.

Plaintext letters that fall in the same column are replaced by the letter beneath, with the top element of the column following the last.

Otherwise, each plaintext letter is replaced by the letter that lies in its own row

And the column occupied by the other plaintext letter.

M	O	N	A	R
C	H	Y	B	D
E	F	G	I/J	K
L	P	Q	S	T
U	V	W	X	Z

Plaintext = meet me at the school house

Splitting two letters as a unit => me et me at th es ch o x ol ho us ex

Corresponding cipher text => CL KL CL RS PD IL HY AV MP HF XL IU

Strength of playfair cipher

Playfair cipher is a great advance over simple mono alphabetic ciphers.

Since there are 26 letters, 26x26 = 676 diagrams are possible, so identification of individual diagram is more difficult.

1.15.1.3 Polyalphabetic ciphers

Another way to improve on the simple monoalphabetic technique is to use different monoalphabetic substitutions as one proceeds through the plaintext message. The general name for this approach is polyalphabetic cipher. All the techniques have the following features in common.

A set of related monoalphabetic substitution rules are used

A key determines which particular rule is chosen for a given transformation.

Vigenere cipher

In this scheme, the set of related monoalphabetic substitution rules consisting of 26 caesar ciphers with shifts of 0 through 25. Each cipher is denoted by a key letter. e.g., Caesar cipher with a shift of 3 is denoted by the key value 'd" (since a=0, b=1, c=2 and so on). To aid in understanding the scheme, a matrix known as vigenere tableau is Constructed

Each of the 26 ciphers is laid out horizontally, with the key letter for each cipher to its left. A normal alphabet for the plaintext runs across the top. The process of

	PLAIN TEXT															
KEY LETTERS		a	b	c	d	e	f	g	h	i	j	k	…	x	y	z
	a	A	B	C	D	E	F	G	H	I	J	K	…	X	Y	Z
	b	B	C	D	E	F	G	H	I	J	K	L	…	Y	Z	A
	c	C	D	E	F	G	H	I	J	K	L	M	…	Z	A	B
	d	D	E	F	G	H	I	J	K	L	M	N	…	A	B	C
	e	E	F	G	H	I	J	K	L	M	N	O	…	B	C	D
	f	F	G	H	I	J	K	L	M	N	O	P	…	C	D	E
	g	G	H	I	J	K	L	M	N	O	P	Q	…	D	E	F
	:	:	:	:	:	:	:	:	:	:	:	:	…	:	:	:
	x	X	Y	Z	A	B	C	D	E	F	G	H	…			W
	y	Y	Z	A	B	C	D	E	F	G	H	I	…			X
	z	Z	A	B	C	D	E	F	G	H	I	J	…			Y

Encryption is simple: Given a key letter X and a plaintext letter y, the cipher text is at the intersection of the row labeled x and the column labeled y; in this case, the ciphertext is V.

To encrypt a message, a key is needed that is as long as the message. Usually, the key is a repeating keyword.

e.g., key = d e c e p t i v e d e c e p t i v e d e c e p t i v e PT = w e a r e d i s c o v e r e d s a v e y o u r s e l f CT = ZICVTWQNGRZGVTWAVZHCQYGLMGJ

Decryption is equally simple. The key letter again identifies the row. The position of the cipher text letter in that row determines the column, and the plaintext letter is at the top of that column.

Strength of Vigenere cipher

- There are multiple cipher text letters for each plaintext letter.
- Letter frequency information is obscured.

One Time Pad Cipher

It is an unbreakable cryptosystem. It represents the message as a sequence of 0s and 1s. this can be accomplished by writing all numbers in binary, for example, or by using ASCII. The key is a random sequence of 0"s and 1"s of same length as the message. Once a key is used, it is discarded and never used again. The system can be expressed as Follows:

$C_i = P_i \oplus K_i$ C_i - i^{th} binary digit of cipher text P_i - i^{th} binary digit of plaintext K_i - i^{th} binary digit of key

Exclusive OR operation

Thus the cipher text is generated by performing the bitwise XOR of the plaintext and the key. Decryption uses the same key. Because of the properties of XOR, decryption simply involves the same bitwise operation:

$P_i = C_i \oplus K_i$

e.g., plaintext = 0 0 1 0 1 0 0 1

Key = 1 0 1 0 1 1 0 0

---------- ciphertext = 1 0 0 0 0 1 0 1

Advantage:

Encryption method is completely unbreakable for a ciphertext only attack.

Disadvantages

It requires a very long key which is expensive to produce and expensive to transmit.

Once a key is used, it is dangerous to reuse it for a second message; any knowledge on the first message would give knowledge of the second.

TRANSPOSITION TECHNIQUES

All the techniques examined so far involve the substitution of a cipher text symbol for a plaintext symbol. A very different kind of mapping is achieved by performing some sort of permutation on the plaintext letters. This technique is referred to as a transposition cipher.

Rail fence

is simplest of such cipher, in which the plaintext is written down as a sequence of diagonals and then read off as a sequence of rows.

Plaintext = meet at the school house

To encipher this message with a rail fence of depth 2, we write the message as follows:

m e a t e c o l o s

e t t h s h o h u e

The encrypted message is

MEATECOLOSETTHSHOHUE

Row Transposition Ciphers-

A more complex scheme is to write the message in a rectangle, row by row, and read the message off, column by column, but permute the order of the columns. The order of columns then becomes the key of the algorithm.

e.g., plaintext = meet at the school house

Key =	4	3	1	2	5	6	7
PT =	m	e	e	t	a	t	t

h e s c h o o

l h o u s e

CT = ESOTCUEEHMHLAHSTOETO

A pure transposition cipher is easily recognized because it has the same letter frequencies as the original plaintext. The transposition cipher can be made significantly more secure by performing more than one stage of transposition. The result is more complex permutation that is not easily reconstructed.

Feistel cipher structure

The input to the encryption algorithm are a plaintext block of length 2w bits and a key K. the plaintext block is divided into two halves L_0 and R_0. The two halves of the data pass through „n" rounds of processing and then combine to produce the ciphertext block. Each round „i" has inputs L_{i-1} and R_{i-1}, derived from the previous round, as well as the subkey K_i, derived from the overall key K. in general, the subkeys K_i are different from K and from each other.

All rounds have the same structure. A substitution is performed on the left half of the data (as similar to S-DES). This is done by applying a round function F to the right half of the data and then taking the XOR of the output of that function and the left half of the data. The round function has the same general structure for each round but is parameterized by the round sub key k_i. Following this substitution, a permutation is performed that consists of the interchange of the two halves of the data. This structure is a particular form of the substitution-permutation network. The exact realization of a Feistel network depends on the choice of the following parameters and design features:

Block size - Increasing size improves security, but slows cipher

Key size - Increasing size improves security, makes exhaustive key searching harder, but may slow cipher

Number of rounds - Increasing number improves security, but slows cipher

Subkey generation - Greater complexity can make analysis harder, but slows cipher

Round function - Greater complexity can make analysis harder, but slows cipher

Fast software en/decryption & ease of analysis - are more recent concerns for practical use and testing.

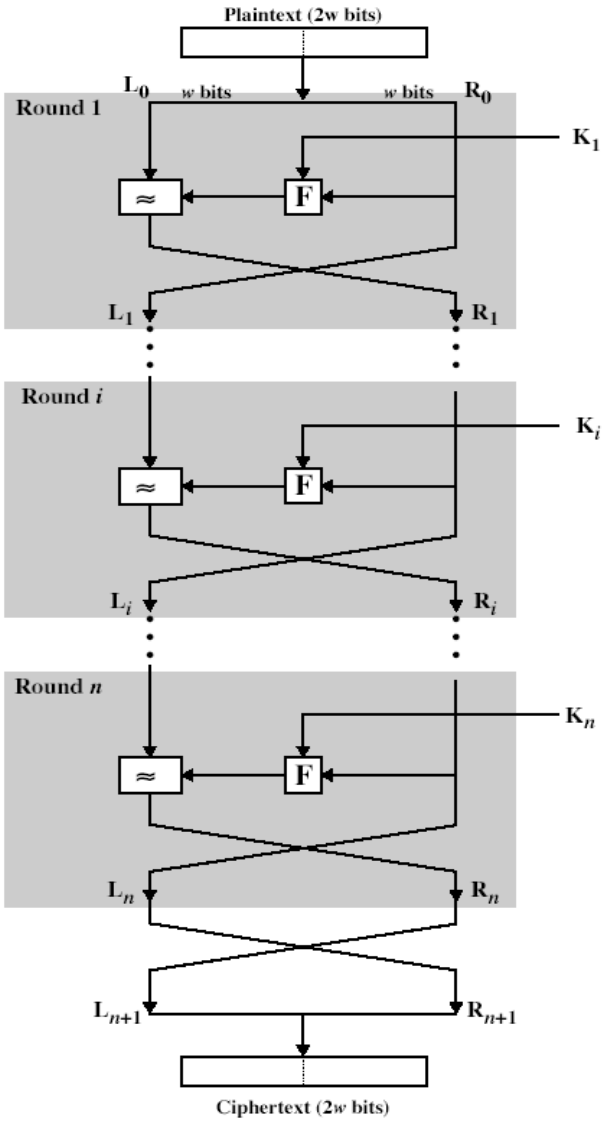


Fig: Classical Feistel Network

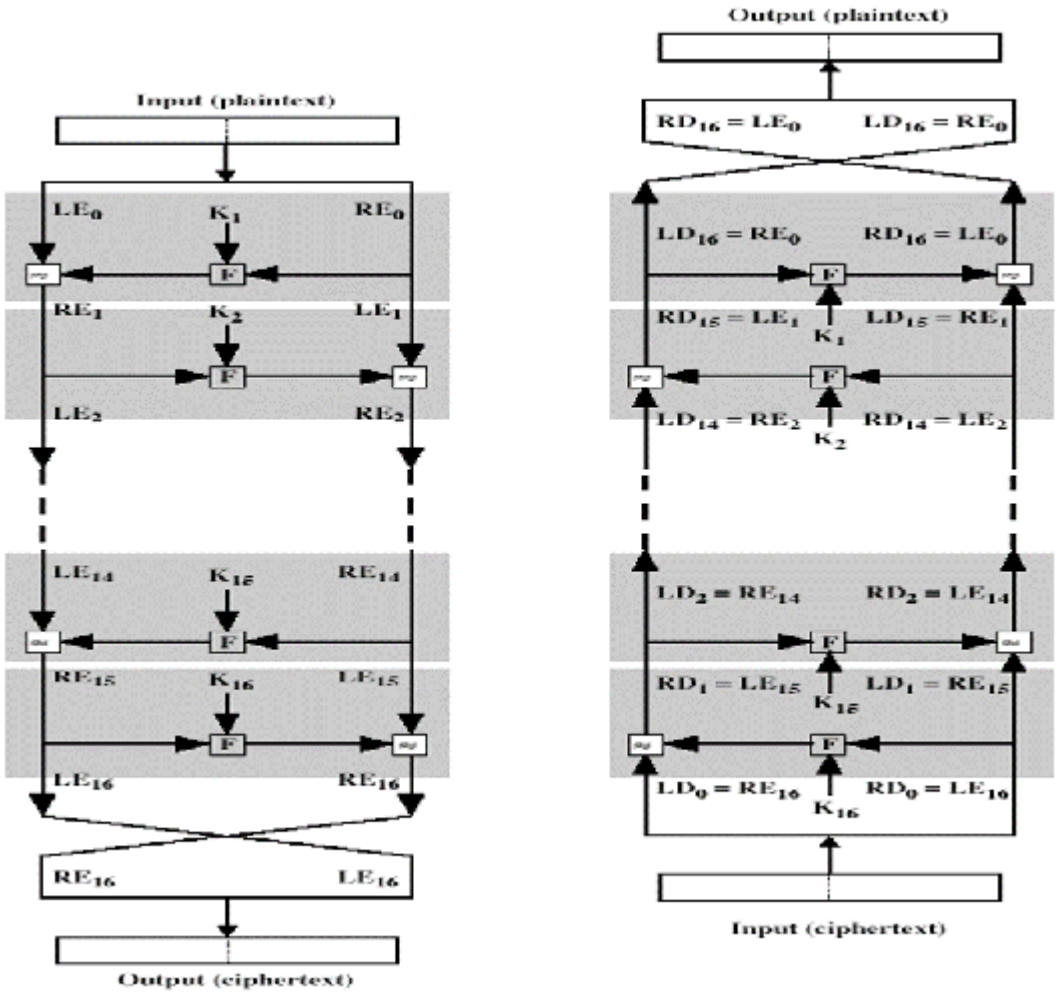


Fig: Feistel encryption and decryption

The process of decryption is essentially the same as the encryption process. The rule is as follows: use the cipher text as input to the algorithm, but use the subkey k_i in reverse order. i.e., k_n in the first round, k_{n-1} in second round and so on. For clarity, we use the notation LE_i and RE_i for data traveling through the decryption algorithm. The diagram below indicates that, at each round, the intermediate value of the decryption process is same (equal) to the corresponding value of the encryption process with two halves of the value swapped.

i.e., $RE_i \| LE_i$ (or) equivalently $RD_{16-i} \| LD_{16-i}$

After the last iteration of the encryption process, the two halves of the output are swapped, so that the cipher text is $RE_{16} \| LE_{16}$. The output of that round is the cipher text. Now take the cipher text and use it as input to the same algorithm. The input to the first round is $RE_{16} \| LE_{16}$, which is equal to the 32-bit swap of the output of the sixteenth round of the encryption process.

Now we will see how the output of the first round of the decryption process is equal to a 32-bit swap of the input to the sixteenth round of the encryption process. First consider the encryption process,

$LE_{16} = RE_{15}$

$RE_{16} = LE_{15} \oplus F(RE_{15}, K_{16})$ On the decryption side,

$LD_1 = RD_0 = LE_{16} = RE_{15}$

$RD_1 = LD_0 \oplus F(RD_0, K_{16})$

$= RE_{16} \oplus F(RE_{15}, K_{16})$

$= [LE_{15} \oplus F(RE_{15}, K_{16})] \oplus F(RE_{15}, K_{16})$

$= LE_{15}$

Therefore, $LD_1 = RE_{15}$

$RD_1 = LE_{15}$ In general, for the i^{th} iteration of the encryption algorithm, $LE_i = RE_{i-1}$

$RE_i = LE_{i-1} \oplus F(RE_{i-1}, K_i)$

Finally, the output of the last round of the decryption process is $RE_0 \| LE_0$. A 32-bit swap recovers the original plaintext.

MODULE - II

BLOCK CIPHER PRINCIPLES

Virtually, all symmetric block encryption algorithms in current use are based on a structure referred to as Fiestel block cipher. For that reason, it is important to examine the design principles of the Fiestel cipher. We begin with a **comparison of stream cipher with block cipher.**

- **A stream cipher** is one that encrypts a digital data stream one bit or one byte at a time. E.g, vigenere cipher. **A block cipher** is one in which a block of plaintext is treated as a whole and used to produce a cipher text block of equal length. Typically a block size of 64 or 128 bits is used.

Block cipher principles

- most symmetric block ciphers are based on a **Feistel Cipher Structure** needed since must be able to **decrypt** ciphertext to recover messages efficiently. block ciphers look like an extremely large substitution
- would need table of 264 entries for a 64-bit block
- Instead create from smaller building blocks
- using idea of a product cipher in 1949 Claude Shannon introduced idea of substitution-permutation (S-P) networks called modern substitution-transposition product cipher
  these form the basis of modern block ciphers

- S-P networks are based on the two primitive cryptographic operations we have seen before:
  - *substitution* (S-box)
  - *permutation* (P-box)
  - provide *confusion* and *diffusion* of message
  - **diffusion** – dissipates statistical structure of plaintext over bulk of ciphertext
- **confusion** – makes relationship between ciphertext and key as complex as possible

DATA ENCRYPTION STANDARD (DES)

In May 1973, and again in Aug 1974 the NBS (now NIST) called for possible encryption algorithms for use in unclassified government applications response was mostly disappointing, however IBM submitted their Lucifer design following a period of redesign and comment it became the Data Encryption Standard (DES)

it was adopted as a (US) federal standard in Nov 76, published by NBS as a hardware only scheme in Jan 77 and by ANSI for both hardware and software standards in ANSI X3.92-1981 (also X3.106-1983 modes of use) subsequently it has been widely adopted and is now published in many standards around the world cf Australian Standard AS2805.5-1985

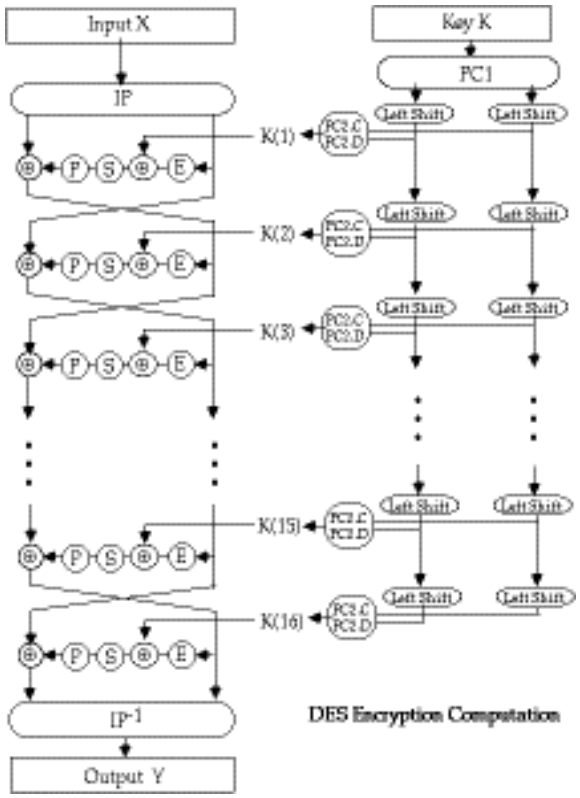
one of the largest users of the DES is the banking industry, particularly with EFT, and EFTPOS

it is for this use that the DES has primarily been standardized, with ANSI having twice reconfirmed its recommended use for 5 year periods - a further extension is not expected however although the standard is public, the design criteria used are classified and have yet to be released there has been considerable controversy over the design, particularly in the choice of a 56-bit key

- recent analysis has shown despite this that the choice was appropriate, and that DES is well designed
- rapid advances in computing speed though have rendered the 56 bit key susceptible to exhaustive key search, as predicted by Diffie & Hellman

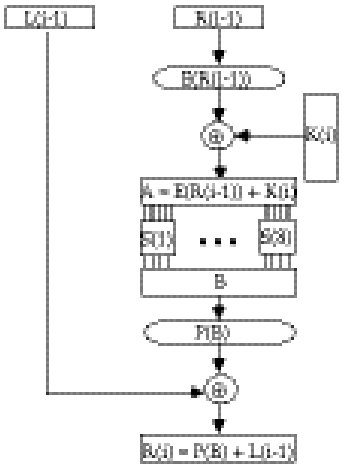
- the DES has also been theoretically broken using a method called Differential Cryptanalysis, however in practice this is unlikely to be a problem (yet)

Overview of the DES Encryption Algorithm



DES Encryption Computation

- the basic process in enciphering a 64-bit data block using the DES consists of:
  - an initial permutation (IP)
  - 16 rounds of a complex key dependent calculation f
  - a final permutation, being the inverse of IP
- in more detail the 16 rounds of f consist of:



- this can be described functionally as

L(i) = R(i-1)

R(i) = L(i-1) (+) P(S(E(R(i-1))(+) K(i)))

and forms one round in an S-P network

- the subkeys used by the 16 rounds are formed by the **key schedule** which consists of:
  - an initial permutation of the key (PC1) which selects 56-bits in two 28-bit halves
  - 16 stages consisting of
  - selecting 24-bits from each half and permuting them by PC2 for use in function f,
  - rotating each half either 1 or 2 places depending on the **key rotation schedule** KS
- this can be described functionally as:

K(i) = PC2(KS(PC1(K),i))

- the **key rotation schedule** KS is specified as:

Round	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
KS	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1
Total Rot	1	2	4	6	8	10	12	14	15	17	19	21	23	25	27	28

- more details on the various DES functions can be found in your textbooks
- following is a walk-through of a DES encryption calculation taken from:

H Katzan, "The Standard Data Encryption Algorithm", Petrocelli Books, New York, 1977

DES Modes of Use

- DES encrypts 64-bit blocks of data, using a 56-bit key
- we need some way of specifying how to use it in practise, given that we usually have an arbitrary amount of information to encrypt
- the way we use a block cipher is called its **Mode of Use** and four have been defined for the DES by ANSI in the standard: ANSI X3.106-1983 Modes of Use)
- modes are either:

Block Modes

Splits messages in blocks (ECB, CBC)

Electronic Codebook Book (ECB)

- Where the message is broken into independent 64-bit blocks which are encrypted

C_(i) = DES_(K1) (P_(i))

Cipher Block Chaining (CBC)

Again the message is broken into 64-bit blocks, but they are linked together in the encryption operation with an IV C_(i) = DES_(K1) (P_(i)(+)C_(i-1)) C_(-1)=IV

Stream Modes

On bit stream messages (CFB, OFB)

Cipher Feedback (CFB)

- Where the message is treated as a stream of bits, added to the output of the DES, with the result being feedback for the next stage
C_(i) = P_(i)(+) DES_(K1) (C_(i-1)) C_(-1)=IV

Output Feedback (OFB)

- Where the message is treated as a stream of bits, added to the message, but with the feedback being independent of the message
C_(i) = P_(i)(+) O_(i) O_(i) = DES_(K1)(O_(i-1)) O_(-1)=IV

- each mode has its advantages and disadvantages

Limitations of Various Modes

ECB

- repetitions in message can be reflected in ciphertext
  - if aligned with message block
  - particularly with data such graphics
  - or with messages that change very little, which become a code-book analysis problem
- weakness is because enciphered message blocks are independent of each other

Block	Plaintext	Ciphertext
1	T H E Y C A N	6D 99 46 42 52 82 22 49
2	H A V E S E	FF BF BC 77 8B BB F2 06
3	V E R A L A C	0D 4D 86 DE B6 CD 92 5D
4	T I V E P E R	99 63 A8 0F 32 D3 E7 E9
5	M A N E N T V	10 49 1F 3B DE 67 21 B7
6	I R T U A L C	BD 2D 6D 61 42 08 C7 B8
7	I R C U I T S	19 F1 01 A4 89 6A AE 4C
8	A N D / O R V	84 DB CC EC 35 1B 5B 9C
9	I R T U A L C	BD 2D 6D 61 42 08 C7 B8
10	A L L S A T	D4 3C D4 5A 9E 0B A5 ED
11	T H E S A M E	84 52 01 AC 2D FE 98 3A
12	T I M E .	89 F1 89 E9 DB CC CB BB
	Key	01 23 45 67 89 AB CD EF

Fig. 4.1 A weakness in ECB encipherment

CBC

- use result of one encryption to modify input of next
  - hence each ciphertext block is dependent on **all** message blocks before it
  - thus a change in the message affects the ciphertext block after the change as well as the original block

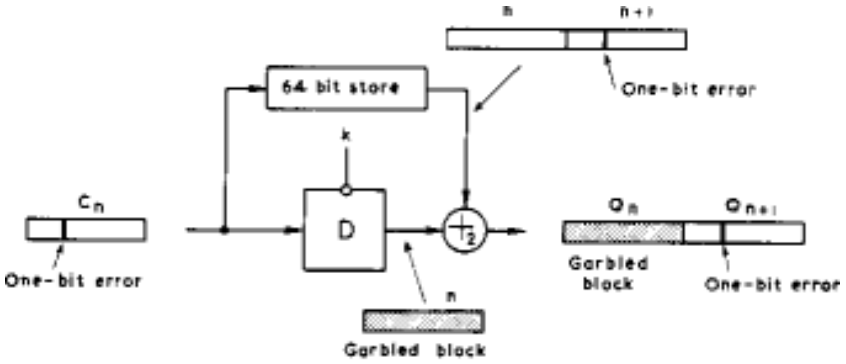


Fig. 4.5 One-bit error in cipher block chaining

to start need an **Initial Value** (IV) which must be known by both sender and receiver

- however if IV is sent in the clear, an attacker can change bits of the first block, and change IV to compensate
- hence either IV must be a fixed value (as in EFTPOS) or it must be sent encrypted in ECB mode before rest of message

- also at the end of the message, have to handle a possible last short block
  - either pad last block (possible with count of pad size), or use some fiddling to double up last two blocks
  - see Davies for examples

CFB

- when data is bit or byte oriented, want to operate on it at that level, so use a stream mode
- the block cipher is use in **encryption** mode at **both** ends, with input being a feed-back copy of the ciphertext
- can vary the number of bits feed back, trading off efficiency for ease of use
- again errors propogate for several blocks after the error

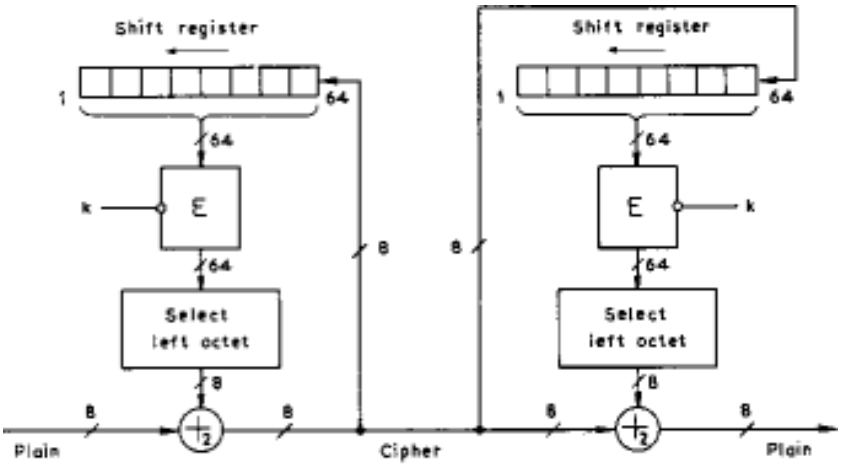


Fig. 4.7 8-bit cipher feedback

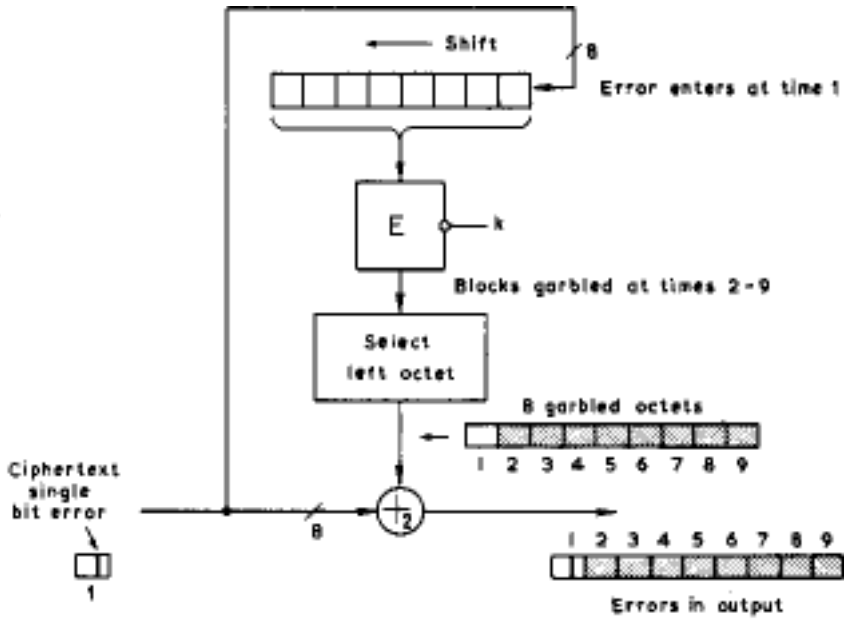


Fig. 4.8 One-bit error in 8-bit cipher feedback

OFB

- also a stream mode, but intended for use where the error feedback is a problem, or where the encryptions want to be done before the message is available
- is superficially similar to CFB, but the feedback is from the output of the block cipher and is independent of the message, a variation of a Vernam cipher
- again an IV is needed
- sender and receiver must remain in sync, and some recovery method is needed to ensure this occurs
- although originally specified with varying m-bit feedback in the standards, subsequent research has shown that only **64-bit OFB** should ever be used (and this is the most efficient use anyway), see

  D Davies, G Parkin, "The Average Cycle Size of the Key Stream in Output Feedback Encipherment" in Advances in Cryptology - Crypto 82, Plenum Press, 1982, pp97-98

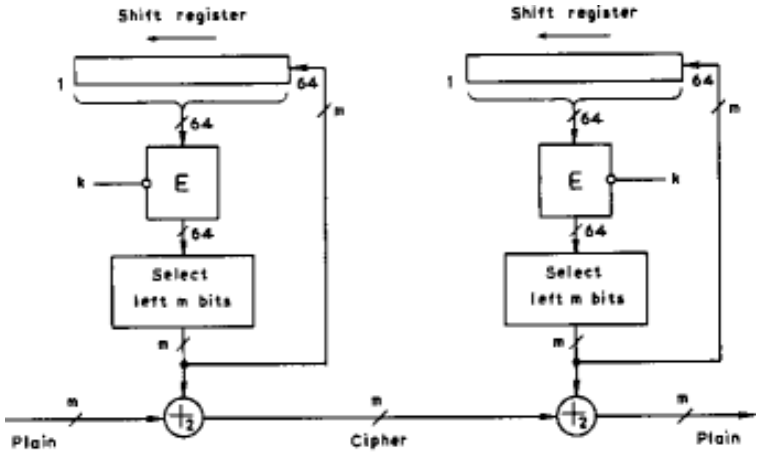


Fig. 4.12 m-bit output feedback

DES Weak Keys

- with many block ciphers there are some keys that should be avoided, because of reduced cipher complexity
- these keys are such that the same sub-key is generated in more than one round, and they include:

Weak Keys

- he same sub-key is generated for every round
- DES has 4 weak keys

Semi-Weak Keys

- only two sub-keys are generated on alternate rounds
- DES has 12 of these (in 6 pairs)

Demi-Semi Weak Keys

- have four sub-keys generated
- none of these cause a problem since they are a tiny fraction of all available keys
- however they MUST be avoided by any key generation program

DES Design Principles

Although the standard for DES is public, the design criteria used are classified and have yet to be released. some information is known, and more has been deduced

L P Brown, "A Proposed Design for an Extended DES", in Computer Security in the Age of Information, W. J. Caelli (ed), North-Holland, pp 9-22, 1989

L P Brown, J R Seberry, "On the Design of Permutation Boxes in DES Type Cryptosystems", in Advances in Cryptology - Eurocrypt '89, Lecture Notes in Computer Science, vol 434, pp 696-705, J.J. Quisquater, J. Vanderwalle (eds), Springer-Verlag, Berlin, 1990.

L P Brown and J R Seberry, "Key Scheduling in DES Type Cryptosystems," in Advances in Cryptology - Auscrypt '90, Lecture Notes in Computer Science, vol 453, pp 221-228, J. Seberry, J. Pieprzyk (eds), Springer-Verlag, Berlin, 1990.

will briefly overview the basic results, for more detailed analyses see the above papers

DES S-Box Design Criteria

Each S-box may be considered as four substitution functions

- these 1-1 functions map inputs 2,3,4,5 onto output bits
- a particular function is selected by bits 1,6
- this provides an **autoclave feature**

DES Design Criteria

- there were 12 criterion used, resulting in about 1000
- possible S-Boxes, of which the implementers chose 8
- these criteria are CLASSIFIED SECRET
- however, some of them have become known
- The following are design criterion:

R1: Each row of an S-box is a permutation of 0 to 15

R2: No S-Box is a linear of affine function of the input

R3: Changing one input bit to an S-box results in changing at least two output bits

R4: S(x) and S(x+001100) must differ in at least 2 bits

- The following are said to be caused by design criteria

R5: S(x) [[pi]] S(x+11*ef*00) for any choice of *e* and *f*

R6: The S-boxes were chosen to minimize the difference between the number of 1's and 0's in any S-box output when any single input is held constant

R7: The S-boxes chosen require significantly more minterms than a random choice would require

Meyer Tables 3-17, 3-18

DES Permutation Tables

- there are 5 Permutations used in DES:
  - **IP** and **IP^(-1) , P, E, PC1, PC2**
- their design criteria are CLASSIFIED SECRET
- it has been noted that **IP** and **IP^(-1)** and **PC1** serve no cryptological function when DES is used in ECB or CBC modes, since searches may be done in the space generated after they have been applied
- **E, P,** and **PC2** combined with the S-Boxes must supply the required dependence of the output bits on the input bits and key bits (**avalanche** and **completeness** effects)

Ciphertext Dependence on Input and Key

- the role of **P, E,** and **PC2** is distribute the outputs of the S-boxes so that each output bit becomes a function of all the input bits in as few rounds as possible
- Carl Meyer (in Meyer 1978, or Meyer & Matyas 1982) performed this analysis on the current DES design

Ciphertext dependence on Plaintext

- define **G_(i,j)** a 64*64 array which shows the dependence of output bits X(j) on input bits X(i)
- examine **G_(0,j)** to determine how fast complete dependence is achieved
- to build **G_(0,1)** use the following

L(i) = R(i-1)

R(i) = L(i-1) (+) f(K(i), R(i-1))

- DES P reaches complete dependence after 5 rounds
- []

Ciphertext dependence on Key

- Carl Meyer also performed this analysis

- define **F_(i,j)** a 64*56 array which shows the dependence of output bits X(j) on key bits U(i) (after PC1 is used)
- examine **F_(0,j)** to determine how fast complete dependence is achieved
- DES PC2 reaches complete dependence after 5 rounds

Key Scheduling and PC2

- Key Schedule
  - is a critical component in the design
  - must provide different keys for each round otherwise security may be compromized (see Grossman & Tuckerman 1978)
  - current scheme can result in weak keys which give the same, 2 or 4 keys over the 16 rounds
- Key Schedule and PC-2 Design
  - is performed in two 28-bit independent halves
  - C-side provides keys to S-boxes 1 to 4
  - D-side provides keys to S-boxes 5 to 8
  - the rotations are used to present different bits of the key for selection on successive rounds
  - PC-2 selects key-bits and distributes them over the S-box inputs

Possible Techniques for Improving DES

- multiple enciphering with DES
- extending DES to 128-bit data paths and 112-bit keys
- extending the Key Expansion calculation

Triple DES

- DES variant
- standardised in ANSI X9.17 & ISO 8732 and in PEM for key management
- proposed for general EFT standard by ANSI X9
- backwards compatible with many DES schemes
- uses 2 or 3 keys

C = DES_(K1) Bbc{(DES^(-1)_(K2)Bbc{(DES_(K1)(P)))

- no known practical attacks

- brute force search impossible
- meet-in-the-middle attacks need 2^(56) PC pairs per key

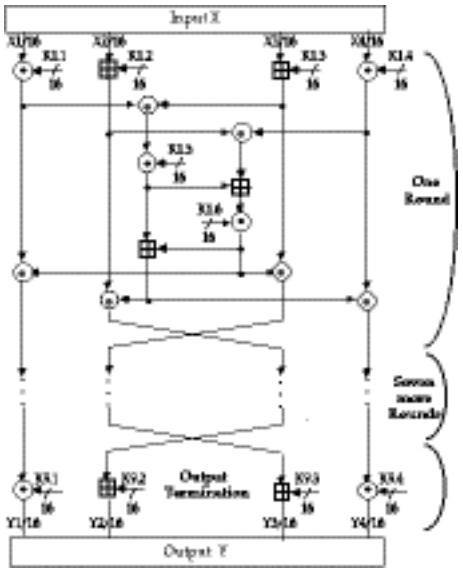
- popular current alternative

IDEA (IPES)

- developed by James Massey & Xuejia Lai at ETH originally in Zurich in 1990, then called IPES :
- Name changed to IDEA in 1992
- encrypts 64-bit blocks using a 128-bit key
- based on mixing operations from different (incompatible) algebraic groups (XOR, Addition mod 2^(16) , Multiplication mod 2^(16) +1)
- all operations are on 16-bit sub-blocks, with no permutations used, hence its very efficient in s/w
- IDEA is patented in Europe & US, however non-commercial use is freely permitted
- used in the public domain PGP secure email system (with agreement from the patent holders)
- currently no attack against IDEA is known (it appears secure against differential cryptanalysis), and its key is too long for exhaustive search

Overview of IDEA

- IDEA encryption works as follows:
  - the 64-bit data block is divided by 4 into: X_(1) , X_(2) , X_(3) , X_(4)
  - in each of eight the sub-blocks are XORd, added, multiplied with one another and with six 16-bit sub-blocks of key material, and the second and third sub-blocks are swapped
  - finally some more key material is combined with the sub-blocks



- IDEA sub-keys
  - the encryption keying material is obtained by splitting the 128-bits of key into eight 16-bit sub-keys, once these are used the key is rotated by 25-bits and broken up again etc
  - the decryption keying material is a little more complex, since inverses of the sub-blocks need to be calculated
- the keys used may be summarised as follows:

Round	Encryption Keys	Decryption Keys
1	K1.1 K1.2 K1.3 K1.4 K1.5 K1.6 K8.6	K9.1-1 -K9.2 -K9.3 K9.4-1 K8.5
2	K2.1 K2.2 K2.3 K2.4 K2.5 K2.6 K7.6	K8.1-1 -K8.3 -K8.2 K8.4-1 K7.5
3	K3.1 K3.2 K3.3 K3.4 K3.5 K3.6 K6.6	K7.1-1 -K7.3 -K7.2 K7.4-1 K6.5
4	K4.1 K4.2 K4.3 K4.4 K4.5 K4.6 K5.6	K6.1-1 -K6.3 -K6.2 K6.4-1 K5.5
5	K5.1 K5.2 K5.3 K5.4 K5.5 K5.6 K4.6	K5.1-1 -K5.3 -K5.2 K5.4-1 K4.5
6	K6.1 K6.2 K6.3 K6.4 K6.5 K6.6 K3.6	K4.1-1 -K4.3 -K4.2 K4.4-1 K3.5
7	K7.1 K7.2 K7.3 K7.4 K7.5 K7.6 K2.6	K3.1-1 -K3.3 -K3.2 K3.4-1 K2.5
8	K8.1 K8.2 K8.3 K8.4 K8.5 K8.6 K1.6	K2.1-1 -K2.3 -K2.2 K2.4-1 K1.5
Output	K9.1 K9.2 K9.3 K9.4	K1.1-1 -K1.2 -K1.3 K1.4-1

where: K1.1^(-1) is the multiplicative inverse mod 2^(16) +1

-K1.2 is the additive inverse mod 2^(16) and the original operations are:

(+) bit-by-bit XOR + additional mod 2^(16) of 16-bit integers

* Multiplication mod 2^(16) +1 (where 0 means 2^(16))

IDEA Example Encryption

Key (128-bits) Plain (64-bit) Cipher (64-bit)
7ca110454a1a6e5701a1d6d039776742 690f5b0d9a26939b 1bddb24214237ec7
idea(X=690f 5b0d 9a26 939b)
r=1, X=690f 5b0d 9a26 939b, SK=7ca1 1045 4a1a 6e57 01a1 d6d0
steps=234a 6b52 e440 840f c70a ef5d 3606 2563 0311 3917 205b e751 5245 bd18
r=2, X=205b e751 5245 bd18, SK=3977 6742 8a94 34dc ae03 43ad
steps=460a 4e93 dcd9 3995 9ad3 7706 d13d 4843 4b2d 1c6a 0d27 97f4 52f9 25ff
r=3, X=0d27 97f4 52f9 25ff, SK=a072 eece 84f9 4220 b95c 0687
steps=3320 86c2 d7f2 7410 e4d2 f2d2 57cb 4a9d 04e4 5caf 37c4 d316 da6d 28bf
r=4, X=37c4 d316 da6d 28bf, SK=5b40 e5dd 9d09 f284 4115 2869
steps=8920 b8f3 7776 69e3 fe56 d110 7266 4376 10c0 8326 99e0 67b6 3bd5 eac5
r=5, X=99e0 67b6 3bd5 eac5, SK=0eb6 81cb bb3a 13e5 0882 2a50
steps=9c69 e981 f70f 8efb 6b66 677a b63b 1db5 f5a8 abe3 69c1 02a7 4262 2518
r=6, X=69c1 02a7 4262 2518, SK=d372 b80d 9776 7427 ca11 0454
steps=d39a bab4 d9d8 75d4 0a42 cf60 ba4a 89aa d175 8bbf 02ef 08ad 310b fe6b
r=7, X=02ef 08ad 310b fe6b, SK=a1a6 e570 1a1d 6d03 4f94 2208
steps=3420 ee1d 4b28 1deb 7f08 f3f6 c124 b51a 04bd c5e1 309d 4f95 2bfc d80a
r=8, X=309d 4f95 2bfc d80a, SK=a943 4dca e034 3ada 072e ece8
steps=3df3 9d5f 0c30 0ada 31c3 9785 44a5 dc2a 7253 b6f8 4fa0 7e63 2ba7 bc22
out, X=4fa0 2ba7 7e63 bc22, SK=1152 869b 95c0 6875
= 1bdd b242 1423 7ec7

Differential Cryptanalysis of Block Ciphers

- Differential Cryptanalysis is a recently (in the public research community) developed method which provides a powerful means of analysing block ciphers
- it has been used to analyse most of the currently proposed block ciphers with varying degrees of success
- usually have a break-even point in number of rounds of the cipher used for which differential cryptanalysis is faster than exhaustive key-space search
- if this number is greater than that specified for the cipher, then it is regarded as broken

Overview of Differential Cryptanalysis

- is a statistical attack against Feistel ciphers
- uses structure in cipher not previously used
- design of S-P networks is such that the output from function *f* is influenced by both input and key

$$R(i)=L(i-1) (+) f(K(i)(+)R(i-1))$$

- hence cannot trace values back through cipher without knowing the values of the key

Biham & Shamir's key idea is to compare two separate encryptions (using the same key) and look at the XOR of the S-box inputs and outputs and this is **independent** of the key being used

$$Ra(i)=f(K(i)(+)Ra(i-1))$$
$$Rb(i)=f(K(i)(+)Rb(i-1))$$

hence

$$Y(i)= Ra(i)(+)Rb(i)$$
$$= f(K(i)(+)Ra(i-1)(+)K(i)(+)Rb(i-1))$$
$$= f(Ra(i-1)(+)Rb(i-1)) = f(X(i))$$

- further various input XOR - output XOR pairs occur with different probabilities
- hence knowing information on these pairs gives us additional information on the cipher

XOR Profiles and Characteristics

- start by compiling a table of input vs output XOR values, an **XOR Profile** for each S-box
- a particular input XOR value and output XOR value pair will occur with some probability
- call such a specified pair, a **characteristic**
- can infer information about key value in one round, if find a pair of encryptions matching a characteristic, and hence knowing input and output XOR values
- have several variant forms of differential cryptanalysis, will discuss just the general form used for attacking many rounds (>8) of a cipher
- can describe 1-round characteristic by:

f(x')->y', Pr(p)
(a',b')->(b',a'(+)f(b')) with prob p

- useful characteristics:

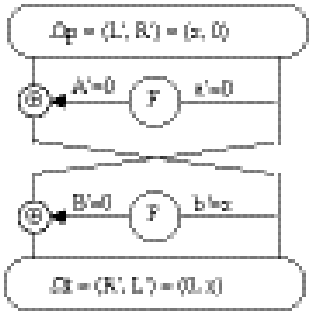
i) f(0')->0', Pr(1) ie always
A.(x,0)->(0,x) always

ii) f(x')->0', Pr(p_(0))

B.(0,x)->(x,0) with probability p_(0)

- attack multiple rounds using **n-round characteristics**
- **n-round characteristics** combine one round characteristics whose outputs & inputs match

- probability of **n-round characteristic** is product of the **1-round characteristic** probabilities



2-Round Iterative Characteristic

- some common characteristic.0000c structures are:

  * a 2-round characteristic:

  A.(x,0)->(0,x) always

  B.(0,x)->(x,0) with probability p

  * a 3-round characteristic:

  A.(x,0)->(0,x) always

  B.(0,x)->(x,x) with probability p1

  C.(x,x)->(x,0) with probability p2

- perform attack by repeatedly encrypting plaintext pairs with known input XOR until obtain expected output XOR matching n-round characteristic being used
- if all intermediate rounds also match required XOR (which is unknown) then have a **right pair**, if not then have a **wrong pair**, relative ratio is S/N for attack
- assume know XOR at intermediate rounds (if right pair) then deduce keys values for the rounds - right pairs suggest same key bits, wrong pairs give random values
- for large numbers of rounds, probability is so low that more pairs are required than exist with 64-bit inputs
- optimisations of this attack can be made, trading memory for search time, and number of rounds used
- in their latest paper, Biham and Shamir show how a 13-round iterated characteristic can be used to break the full 16-round DES

Linear Cryptanalysis of Block Ciphers

- Linear Cryptanalysis is another recently developed method for analysing block ciphers
- like differential cryptanalysis it is a statistical method

- again have a break-even point in number of rounds of the cipher used for which linear cryptanalysis is faster than exhaustive key-space search
- if this number is greater than that specified for the cipher, then it is regarded as broken
- In Linear Cryptanalysis want to find a linear approximation which holds with Prob p!=^(1) /_(2)

  P[i1,i2,...,ia](+)C[j1,j2,...,jb]=K[k1,k2,...,kc]
  where ia,jb,kc are bit locations in P,C,K
- can determine one bit of key using maximum likelihood algorithm, using a large number of trial encryptions
- effectiveness of linear cryptanalysis is given by

  |p - 1/2|
- DES can be broken by encrypting 2^(47) known plaintexts

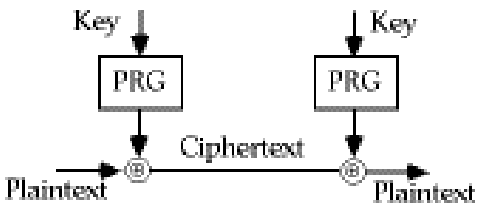
PL[7,18,24](+) PR[12,16](+) CL[15](+) CR[7,18,24,29](+) F16(CR,K16)[15] = K1[19,23](+)K3[22](+) K4[44](+) K5[22](+)K7[22](+) K8[44](+) K9[22](+) K11[22](+) K12[44](+) K13[22](+) K15[22]

- this will recover some of the key bits, the rest must be searched for exhaustively
- LOKI with 12 or more rounds cannot be broken using linear cryptanalysis

Stream Ciphers and the Vernam cipher

- Process the message bit by bit (as a stream)
- The most famous of these is the **Vernam cipher** (also known as the **one-time pad**)
- invented by Vernam, working for AT&T, in 1917
- simply add bits of message to random key bits
- need as many key bits as message, difficult in practise (ie distribute on a mag-tape or CDROM)
- is unconditionally secure provided key is truly random

- suggest generating keystream from a smaller (base) key



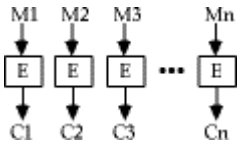
- use some pseudo-random function to do this

Modern Private Key Ciphers (part 1)

- now want to concentrate on modern encryption systems
- these usually consider the message as a sequence of bits
  - (eg as a series of ASCII characters concatenated)
- have two broad families of methods
  - stream ciphers and block ciphers

Block Ciphers

- in a block cipher the message is broken into blocks, each of which is then encrypted (ie like a substitution on very big characters - 64-bits or more)
- most modern ciphers we will study are of this form



Shannons Theory of Secrecy Systems

- Claude Shannon wrote some of the pivotal papers on modern cryptology theory in 1949:
  - C E Shannon, "Communication Theory of Secrecy Systems", Bell System Technical Journal, Vol 28, Oct 1949, pp 656-715
  - C E Shannon, "Prediction and Entropy of printed English", Bell System Technical Journal, Vol 30, Jan 1951, pp 50-64
- in these he developed the concepts of:
  - entropy of a message,
  - redundancy in a language,
  - theories about how much information is needed to break a cipher
  - defined the concepts of computationally secure vs unconditionally secure ciphers
- he showed that the Vernam cipher is the only currently known unconditionally secure cipher, provided the key is truly random
- also showed that if try to encrypt English text by adding to other English text (ie a **Book**cipher), this is not secure since English is 80% redundant, giving ciphertext with 60% redundancy, enough to break

- a similar technique can also be used if the same random key stream is used twice on different messages, the redundancy in the messages is sufficient to break this
- as discussed earlier, **exhaustive key search** is the most fundamental attack, and is directly proportional to the size of the key
- can tabulate these for reasonable assumptions about the number of operations possible (& parallel tests):

Key Size (bits)	Time (1us/test)	Time (1us/106test)
24	8.4 sec	8.4 usec
32	35.8 mins	2.15 msec
40	6.4 days	550 msec
48	4.46 yrs	2.35 mins
56	~2000 yrs	10.0 hrs
64	~500000 yrs	107 days

- as the ultimate limit, it can be shown from energy consumption considerations that the maximum number of possible elementary operations in 1000 years is about: 3 x 10 ^(48)
- similarly can show that if need say 10 atoms to store a bit of information, then the greatest possible number of bits storable in a volume of say the moon is: 10 ^(45)
- if a cipher requires more operations, or needs more storage than this, it is pretty reasonable to say it is computationally secure
  - eg to test all possible 128-bit keys in Lucifer takes about 3 x 10 ^(48) encryptions, needing 10 ^(19) years

Substitution-Permutation Ciphers

- in his 1949 paper Shannon also introduced the idea of substitution-permutation (S-P) networks, which now form the basis of modern block ciphers
- an S-P network is the modern form of a substitution-transposition product cipher
- S-P networks are based on the two primitive cryptographic operations we have seen before

Substitution Operation

- a binary word is replaced by some other binary word
- the whole substitution function forms the key
- if use n bit words, the key is 2^(n)!bits, grows rapidly

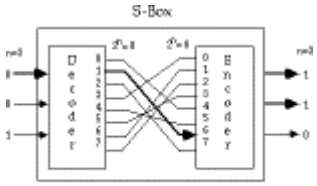


Fig 2.1 Substitution Operation

- can also think of this as a large lookup table, with n address lines (hence 2^(n) addresses), each n bits wide being the output value
- will call them **S-boxes**

Permutation Operation

- a binary word has its bits reordered (permuted)
- the re-ordering forms the key
- if use n bit words, the key is n!bits, which grows more slowly, and hence is less secure than substitution

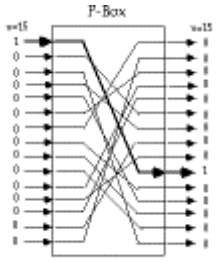


Fig 2.2 - Permutation or Transposition Function

- this is equivalent to a wire-crossing in practise (though is much harder to do in software)
- will call these **P-boxes**

Substitution-Permutation Network

- Shannon combined these two primitives
- he called these **mixing transformations**

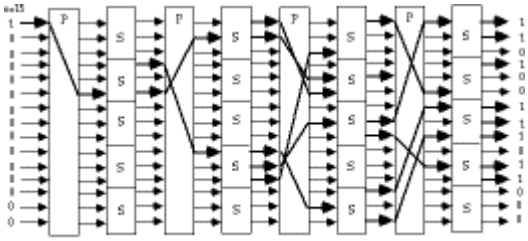


Fig 2.3 - Substitution-Permutation Network, with the Avalanche Characteristic

- Shannons mixing transformations are a special form of product ciphers where

S-Boxes provide **confusion** of input bits
P-Boxes provide **diffusion** across S-box inputs

- in general these provide the following results, as described in:

A F Webster & S E Tavares "On the Design of S-boxes", in Advances in Cryptology - Crypto 85, Lecture Notes in Computer Science, No 218, Springer-Verlag, 1985, pp 523-534

Avalanche effect

- where changing **one** input bit results in changes of approx **half** the output bits

More formally, a function f has a good **avalanche** effect if for each bit i,0<=i<m, if the 2^(m) plaintext vectors are divided into 2^(m-1) pairs X and X_(i) with each pair differing only in bit i; and if the 2^(m-1) exclusive-or sums, termed avalanche vectors

V_(i) = f(X) (+) f(X_(i))

Are compared, then about half of these sums should be found to be 1.

Completeness effect

- where each output bit is a complex function of **all** the input bits

More formally, a function f has a good **completeness** effect if for each bit j,0<=j<m, in the ciphertext output vector, there is at least one pair of plaintext vectors X and X_(i) which differ only in bit i, and for which f(X) and f(X_(i)) differ in bit j

Practical Substitution-Permutation Networks

- in practise we need to be able to decrypt messages, as well as to encrypt them, hence either:
  - have to define inverses for each of our S & P-boxes, but this doubles the code/hardware needed, or
  - define a structure that is easy to reverse, so can use basically the same code or hardware for both encryption and decryption
- Horst Feistel, working at IBM Thomas J Watson Research Labs devised just such a structure in early 70's, which we now call a **feistel cipher**
  - the idea is to partition the input block into two halves, L(i-1)and R(i-1), and use only R(i-1)in each round i (part) of the cipher
  - the function g incorporates one stage of the S-P network, controlled by part of the key K(i)known as the ith subkey

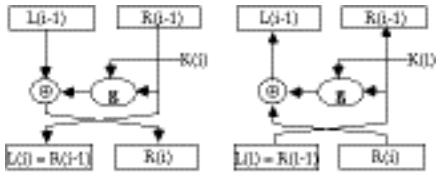


Fig 2.4 - A Round of a Feistel Cipher

- this can be described functionally as:

L(i) = R(i-1)

R(i) = L(i-1) (+) g(K(i), R(i-1))

- this can easily be reversed as seen in the above diagram, working backwards through the rounds
- in practise link a number of these stages together (typically 16 rounds) to form the full cipher

MODULE - III

Modular Arithmetic

Modular arithmetic is 'clock arithmetic' a **congruence** a = b mod n says when divided by *n* that a and b have the same remainder

100 = 34 mod 11

usually have 0<=b<=n-1

-12mod7 = -5mod7 = 2mod7 = 9mod7

b is called the **residue** of a mod n

can do arithmetic with integers modulo n with all results between 0 and n

Addition
a+b mod n

Subtraction

a-b mod n = a+(-b) mod n

Multiplication

a.b mod n

- derived from repeated addition
- can get a.b=0 where neither a,b=0
  - eg 2.5 mod 10

Division
a/b mod n

- is multiplication by inverse of b: $a/b = a.b^{-1} \bmod n$
- if n is prime $b^{-1} \bmod n$ exists s.t $b.b^{-1} = 1 \bmod n$
  - eg 2.3=1 mod 5 hence 4/2=4.3=2 mod 5
- integers modulo n with addition and multiplication form a commutative ring with the laws of

Associativity
(a+b)+c = a+(b+c) mod n

Commutativity

a+b = b+a mod n

Distributivity

$(a+b).c = (a.c)+(b.c) \bmod n$

- also can chose whether to do an operation and then reduce modulo n, or reduce then do the operation, since reduction is a homomorphism from the ring of integers to the ring of integers modulo n
  - $a+/-b \bmod n = [a \bmod n +/- b \bmod n] \bmod n$
  - (the above laws also hold for multiplication)
- if n is constrained to be a prime number *p* then this forms a **Galois Field modulo p** denoted **GF(p)** and all the normal laws associated with integer arithmetic work

Exponentiation in GF(p)

- many encryption algorithms use exponentiation - raising a number a (base) to some power b (exponent) mod p
  - $b = a^e \bmod p$
- exponentiation is basically repeated multiplication, which take s O(n) multiples for a number n
- a better method is the square and multiply algorithm

let base = a, result =1
for each bit ei (LSB to MSB) of exponent
if ei=0 then
square base mod p
if ei=1 then
multiply result by base mod p
square base mod p (except for MSB)
required ae is result

- only takes $O(\log_2 n)$ multiples for a number n

see Sebbery p9 Fig2.1 + example

Discrete Logarithms in GF(p)

- the inverse problem to exponentiation is that of finding the **discrete logarithm** of a number modulo p
  - find x where $a^x = b \bmod p$

Seberry examples p10

- whilst exponentiation is relatively easy, finding discrete logarithms is generally a **hard** problem, with no easy way

- in this problem, we can show that if p is prime, then there always exists an a such that there is always a discrete logarithm for any b!=0
  - successive powers of a "generate" the group mod p
- such an a is called a **primitive root** and these are also relatively hard to find

2.1.3 Greatest Common Divisor

- the greatest common divisor (a,b) of a and b is the largest number that divides evenly into both a and b
- **Euclid's Algorithm** is used to find the Greatest Common Divisor (GCD) of two numbers a and n, a<n
  - use fact if a and b have divisor d so does a-b, a-2b

GCD (a,n) is given by:
let g0=n
g1=a
gi+1 = gi-1 mod gi
when gi=0 then (a,n) = gi-1
eg find (56,98)

g0=98
g1=56
g2 = 98 mod 56 = 42
g3 = 56 mod 42 = 14
g4 = 42 mod 14 = 0
hence (56,98)=14

Inverses and Euclid's Extended GCD Routine

- unlike normal integer arithmetic, sometimes a number in modular arithmetic has a unique inverse
  - a^{-1} is inverse of a mod n if $a.a^{-1} = 1 \bmod n$
  - where a,x in {0,n-1}
  - eg $3.7 = 1 \bmod 10$
- if (a,n)=1 then the inverse always exists
- can extend **Euclid's Algorithm** to find Inverse by keeping track of $g_i = u_i.n + v_i.a$
- **Extended Euclid's** (or Binary GCD) **Algorithm** to find Inverse of a number a mod n (where (a,n)=1) is:

Inverse(a,n) is given by:
g0=n u0=1 v0=0
g1=a u1=0 v1=1

let
y = gi-1 div gi
gi+1 = gi-1 - y.gi = gi-1 mod gi
ui+1 = ui-1 - y.ui
vi+1 = vi-1 - y.vi
when gi=0 then Inverse(a,n) = vi-1

Example

eg: want to find Inverse(3,460):

i	y	g	u	v
0	-	460	1	0
1	-	3	0	1
2	153	1	1	-153
3	3	0	-3	460

hence Inverse(3,460) = -153 = 307 mod 460

Euler Totient Function [[phi]](n)

- if consider arithmetic modulo n, then a **reduced set of residues** is a subset of the complete set of residues modulo n which are relatively prime to n
  - eg for n=10,
  - the complete set of residues is {0,1,2,3,4,5,6,7,8,9}
  - the reduced set of residues is {1,3,7,9}
- the number of elements in the reduced set of residues is called the **Euler Totient function [[phi]](n)**
- there is no single formula for [[phi]](n) but for various cases count how many elements are excluded[4]:

p (p prime)	[[phi]](p) =p-1
pr (p prime)	[[phi]](p) =pr-1(p-1)
p.q (p,q prime)	[[phi]](p.q) =(p-1)(q-1)

see Seberry Table 2.1 p13

- several important results based on [[phi]](n) are:
- **Theorem (Euler's Generalization)**
  - let gcd(a,n)=1 then
  - $a^{[[phi]](n)} \bmod n = 1$
- Fermat's Theorem

- let p be a prime and gcd(a,p)=1 then
- $a^{p-1} \bmod p = 1$

- Algorithms to find **Inverses** $a^{-1} \bmod n$

1. search 1,...,n-1 until an a^{-1} is found with $a.a^{-1} \bmod n$
2. if [[phi]](n) is known, then from Euler's Generalization
   - $a^{-1} = a^{[[phi]](n)-1} \bmod n$
3. otherwise use Extended Euclid's algorithm for inverse

Computing with Polynomials in GF(qn)

- have seen arithmetic modulo a prime number GF(p)
- also can do arithmetic modulo q over polynomials of degree n, which also form a **Galois Field GF(q^n)**
- its elements are polynomials of degree (n-1) or lower
  - $a(x)=a_{n-1}x^{n-1}+a_{n-2}x^{n-2}+...+a_1x+a_0$
- have residues for polynomials just as for integers
  - $p(x)=q(x)d(x)+r(x)$
  - and this is unique if deg[r(x)]<deg[d(x)]
- if r(x)=0, then d(x) **divides** p(x), or is a **factor** of p(x)
- addition in GF(q^n) just involves summing equivalent terms in the polynomial modulo q (XOR if q=2)
  - $a(x)+b(x)=(a_{n-1}+b_{n-1})x^{n-1}+...+(a_1+b_1)x+(a_0+b_0)$

Multiplication with Polynomials in GF(qn)

- multiplication in GF(q^n) involves [5]
  - multiplying the two polynomials together (cf longhand multiplication; here use shifts & XORs if q=2)
  - then finding the residue modulo a given **irreducible polynomial** of degree n
- an **irreducible polynomial** d(x) is a 'prime' polynomial, it has no polynomial divisors other than itself and 1
- modulo reduction of p(x) consists of finding some r(x) st: p(x)=q(x)d(x)+r(x)
  - nb. in GF(2^n) with $d(x)=x^3+x+1$ can do simply by replacing x^3 with x+1

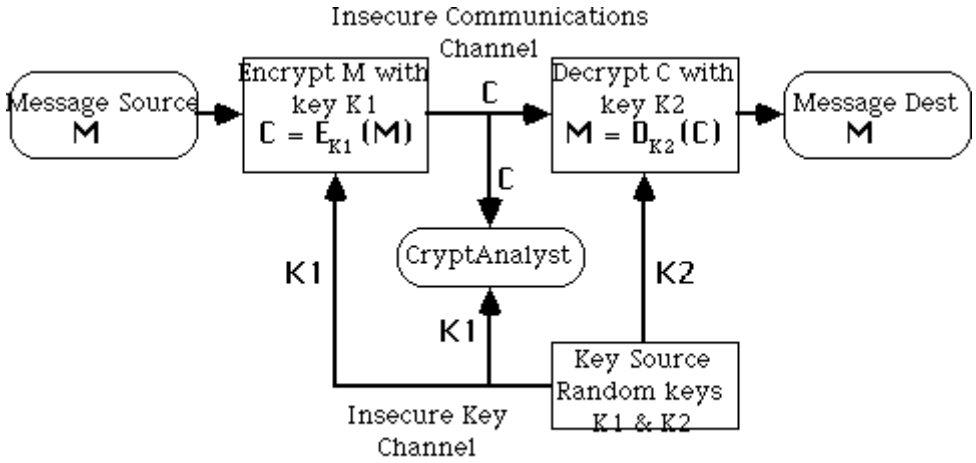
- eg in GF(2^3) there are 8 elements:
  - 0, 1, x, x+1, x^2, x^2+1, x^2+x, x^2+x+1
- with irreducible polynomial d(x)=x^3+x+1* arithmetic in this field can be summarised as:

Seberry Table 2.3 p20

- can adapt GCD, Inverse, and CRT algorithms for GF(q^n)
  - [[phi]](p(x)) = 2^n-1 since every poly except 0 is relatively prime to p(x)
- arithmetic in GF(q^n) can be much faster than integer arithmetic, especially if the irreducible polynomial is carefully chosen
  - eg a fast implementation of GF(2^{127}) exists
- has both advantages and disadvantages for cryptography, calculations are faster, as are methods for breaking

Public-Key Ciphers

- traditional **secret key** cryptography uses a single key shared by both sender and receiver
- if this key is disclosed communications are compromised
- also does not protect sender from receiver forging a message & claiming is sent by sender, parties are equal
- **public-key** (or **two-key**) **cryptography** involves the use of two keys:
  - a **public-key**, which may be known by anybody, and can be used to **encrypt messages**, and **verify signatures**
  - a **private-key**, known only to the recipient, used to **decrypt messages**, and **sign** (create) **signatures**



Asymmetric (Public-Key) Encryption System

- the public-key is easily computed from the private key and other information about the cipher (a polynomial time (P-time) problem)
- however, knowing the public-key and public description of the cipher, it is still computationally infeasible to compute the private key (an NP-time problem)
- thus the public-key may be distributed to anyone wishing to communicate securly with its owner (although secure distribution of the public-key is a non-trivial problem - the **key distribution** problem)
- have three important classes of public-key algorithms:
  - **Public-Key Distribution Schemes** (PKDS) - where the scheme is used to securely exchange a single piece of information (whose value depends on the two parties, but cannot be set).
  - This value is normally used as a session key for a private-key scheme
  - **Signature Schemes** - used to create a digital signature only, where the private-key signs (create) signatures, and the public-key verifies signatures
  - **Public Key Schemes (PKS)** - used for encryption, where the public-key encrypts messages, and the private-key decrypts messages.
  - Any public-key scheme can be used as a PKDS, just by selecting a message which is the required session key
  - Many public-key schemes are also signature schemes (provided encryption& decryption can be done in either order)

RSA Public-Key Cryptosystem

- best known and widely regarded as most practical public-key scheme was proposed by Rivest, Shamir & Adleman in 1977:

R L Rivest, A Shamir, L Adleman, "On Digital Signatures and Public Key Cryptosystems", Communications of the ACM, vol 21 no 2, pp120-126, Feb 1978

- it is a public-key scheme which may be used for encrypting messages, exchanging keys, and creating digital signatures
- is based on exponentiation in a finite (Galois) field over integers modulo a prime
  - nb exponentiation takes $O((\log n)^3)$ operations
- its security relies on the difficulty of calculating factors of large numbers
  - nb factorization takes $O(e^{\log n \log \log n})$ operations
  - (same as for discrete logarithms)
- the algorithm is patented in North America (although algorithms cannot be patented elsewhere in the world)
  - this is a source of legal difficulties in using the scheme

- RSA is a public key encryption algorithm based on exponentiation using modular arithmetic
- to use the scheme, first generate keys:
- Key-Generation by each user consists of:
  - selecting two large primes at random (~100 digit), p, q
  - calculating the system modulus R=p.q p, q primes
  - selecting at random the encryption key e,
  - e < R, gcd(e, F(R)) = 1
  - solving the congruence to find the decryption key d,
  - e.d [[equivalence]] 1 mod [[phi]](R) 0 <= d <= R
  - publishing the public encryption key: K1={e,R}
  - securing the private decryption key: K2={d,p,q}
- Encryption of a message M to obtain ciphertext C is:
- $C = M^e \bmod R$ 0 <= d <= R
- Decryption of a ciphertext C to recover the message M is:
  - $M = C^d = M^{e.d} = M^{1+n.[[phi]](R)} = M \bmod R$
- the RSA system is based on the following result:

if R = pq where p, q are distinct large primes then
X [[phi]](R) = 1 mod R
for all x not divisible by p or q
and [[Phi]](R) = (p-1)(q-1)

RSA Example

- usually the encryption key e is a small number, which must be relatively prime to [[phi]](R) (ie GCD(e, [[phi]](R)) = 1)
- typically e may be the same for all users (provided certain precautions are taken), 3 is suggested
- the decryption key d is found by solving the congruence:

e.d [[equivalence]] 1 mod [[phi]](R), 0 <= d <= R,

- an extended Euclid's GCD or Binary GCD calculation is done to do this.

given e=3, R=11*47=517, [[phi]](R)=10*46=460
then d=Inverse(3,460) by Euclid's alg:

i	y	g	u	v
0	-	460	1	0
1	-	3	0	1

2 153 1 1 -153
3 3 0 -3 460
ie: d = -153, or 307 mod 517

- a sample RSA encryption/decryption calculation is:

M = 26
C = 263 mod 517 = 515
M = 515307 mod 517 = 26

-

Security of RSA

- The security of the RSA scheme rests on the difficulty of factoring the modulus of the scheme R

- best known factorization algorithm (Brent-Pollard) takes:

$$O\left(\frac{e^{\sqrt{2 \ln p \ln \ln p}}}{\ln p}\right)$$

operations on number R whose largest prime factor is p

Decimal Digits in R	#Bit Operations to Factor R
20	7200
40	3.11e+06
60	4.63e+08
80	3.72e+10
100	1.97e+12
120	7.69e+13
140	2.35e+15
160	5.92e+16
180	1.26e+18
200	2.36e+19

- This leads to R having a length of 200 digits (or 600 bits) given that modern computers perform 1-100 MIPS the above can be divided by 10^6 to get a time in seconds

  - nb: currently 1e+14 operations is regarded as a limit for computational feasability and there are 3e+13 usec/year

- but most (all!!) computers can't directly handle numbers larger than 32-bits (64-bits on the very newest)

- hence need to use **multiple precision arithmetic** libraries to handle numbers this large

Multi-Precision Arithmetic

- involves libraries of functions that work on multiword (multiple precision) numbers

- classic references are in Knuth vol 2 - "Seminumerical Algorithms"

- multiplication digit by digit
- do exponentiation using square and multiply[6]

- are a number of well known multiple precision libraries available - so don't reinvent the wheel!!!!
- can use special tricks when doing modulo arithmetic, especially with the modulo reductions

Faster Modulo Reduction

* Chivers (1984) noted a fast way of performing modulo reductions whilst doing multi-precision arithmetic calcs

Given an integer A of n characters (a_0, ... , a_{n-1}) of base b

$$A = \sum_{i=0}^{n-1} a_i b^i \text{ then}$$

$$A \equiv \left\{ \sum_{i=0}^{n-2} a_i b^i + a_{n-1} b^{n-1} \pmod{jm} \right\} \pmod{m}$$

ie: this implies that the MSD of a number can be removed and its remainder mod m added to the remaining digits will result in a number that is congruent mod m to the original.

* Chivers algorithm for reducing a number is thus:

1. Construct an array **R = (b^d, $2.b^d$, ... , $(b-1).b^d$)(mod m)**

2. FOR i = n-1 to d do

WHILE A[i] != 0 do

j = A[i];

A[i] = 0;

A = A + b^{i-d}.R[j];

END WHILE

END FOR

where A[i] is the i^{th} character of number A

R[j] is the j^{th} integer residue from the array R

n is the number of symbols in A

d is the number of symbols in the modulus

Speeding up RSA - Alternate Multiplication Techniques

- conventional multiplication takes $O(n^2)$ bit operations, faster techniques include:
- the Schonhage-Strassen Integer Multiplication Algorithm:
  - breaks each integer into blocks, and uses them as coefficients of a polynomial
  - evaluates these polynomials at suitable points, & multiplies the resultant values
  - interpolates these values to form the coefficients of the product polynomial
  - combines the coefficients to form the product of the original integer
  - the Discrete Fourier Transform, and the Convolution Theorem are used to speed up the interpolation stage
  - can multiply in O(n log n) bit operations
- the use of specialized hardware because:
  - conventional arithmetic units don't scale up, due to carry propogation delays
  - so can use serial-parallel carry-save, or delayed carry-save techniques with O(n) gates to multiply in O(n) bit operations,
  - or can use parallel-parallel techniques with $O(n^2)$ gates to multiply in O(log n) bit operations

RSA and the Chinese Remainder Theorem

- a significant improvement in decryption speed for RSA can be obtained by using the Chinese Remainder theorem to work modulo p and q respectively
  - since p,q are only half the size of R=p.q and thus the arithmetic is much faster
- CRT is used in RSA by creating two equations from the decryption calculation:

$$M = Cd \bmod R$$

as follows:

$$M1 = M \bmod p = (C \bmod p)d \bmod (p-1)$$
$$M2 = M \bmod q = (C \bmod q)d \bmod (q-1)$$

then the pair of equations

$$M = M1 \bmod p \qquad M = M2 \bmod q$$

has a unique solution by the CRT, given by:

$$M = [((M2 + q - M1)u \bmod q] p + M1$$

where

$$p.u \bmod q = 1$$

Primality Testing and RSA

- The first stage of key-generation for RSA involves finding two large primes p, q

- Because of the size of numbers used, must find primes by trial and error
- Modern primality tests utilize properties of primes eg:
  - $a^{n-1} = 1 \bmod n$ where GCD(a,n)=1
  - all primes numbers 'n' will satisfy this equation
  - some composite numbers will also satisfy the equation, and are called pseudo-primes.
- Most modern tests guess at a prime number 'n', then take a large number (eg 100) of numbers 'a', and apply this test to each. If it fails the number is composite, otherwise it is is probably prime.
- There are a number of stronger tests which will accept fewer composites as prime than the above test. eg:

$$\text{GCD}(a,n) = 1, \quad \text{and} \quad \left(\frac{a}{n}\right) (\text{mod } n) = a^{\frac{(n-1)}{2}} (\text{mod } n)$$

$$\text{where } \left(\frac{a}{n}\right) \text{ is the Jacobi symbol}$$

RSA Implementation in Practice

- Software implementations
  - generally perform at 1-10 bits/second on block sizes of 256-512 bits
  - two main types of implementations:
    - - on micros as part of a key exchange mechanism in a hybrid scheme
    - - on larger machines as components of a secure mail system
- Harware Implementations
  - generally perform 100-10000 bits/sec on blocks sizes of 256-512 bits
  - all known implementations are large bit length conventional ALU units

ElGamal

- A variant of the Diffie-Hellman key distribution scheme, allowing secure exchange of messages
- published in 1985 by ElGamal in

T. ElGamal, "A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms", IEEE Trans. Information Theory, vol IT-31(4), pp469-472, July 1985.

- like Diffie-Hellman its security depends on the difficulty of factoring logarithms

- **Key Generation**
  - select a large prime p (~200 digit), and
  - [[alpha]] a primitive element mod p
  - A has a secret number x_A
  - B has a secret number x_B
  - A and B compute y_A and y_B respectively, which are then made public
    - $y_A = [[alpha]]^{xA} \bmod p$
    - $y_B = [[alpha]]^{xB} \bmod p$
- to **encrypt** a message **M** into ciphertext **C**,
  - selects a random number **k,** 0 <= k <= p-1
  - computes the message key **K**
    - $K = y_B^k \bmod p$
  - computes the ciphertext pair: C = {c1,c2}
    - $C_1 = [[alpha]]^k \bmod p$ $C_2 = K.M \bmod p$
- to **decrypt** the message
  - extracts the message key **K**
    - $K = C_1^{xB} \bmod p = [[alpha]]^{k.xB} \bmod p$
  - extracts **M** by solving for M in the following equation:
    - $C_2 = K.M \bmod p$

Other Public-Key Schemes

- a number of other public-key schemes have been proposed, some of the better known being:
  - Knapsack based schemes
  - McEleice's Error Correcting Code based schems
- ALL of these schemes have been broken
- the only currently known secure public key schemes are those based on exponentiation (all of which are patented in North America)
- it has proved to be very difficult to develop secure public key schemes
- this in part is why they have not been adopted faster, as their theorectical advantages might have suggested

AUTHENTICATION REQUIREMENTS

In the context of communication across a network, the following attacks can be identified:

Disclosure – releases of message contents to any person or process not possessing the appropriate cryptographic key.

Traffic analysis – discovery of the pattern of traffic between parties.

Masquerade – insertion of messages into the network fraudulent source.

Content modification – changes to the content of the message, including insertion deletion, transposition and modification.

Sequence modification – any modification to a sequence of messages between parties, including insertion, deletion and reordering.

Timing modification – delay or replay of messages.

Source repudiation – denial of transmission of message by source.

Destination repudiation – denial of transmission of message by destination.

easures to deal with first two attacks are in the realm of message confidentiality. Measures to deal with 3 through 6 are regarded as message authentication. Item 7 comes under digital signature and dealing with item 8 may require a combination of digital signature and a protocol to counter this attack.

AUTHENTICATION FUNCTIONS

Any message authentication or digital signature mechanism can be viewed as having fundamentally two levels. At the lower level, there may be some sort of function that produces an authenticator: a value to be used to authenticate a message. This lower layer function is then used as primitive in a higher-layer authentication protocol that enables a receiver to verify the authenticity of a message.

The different types of functions that may be used to produce an **authenticator** are as follows:

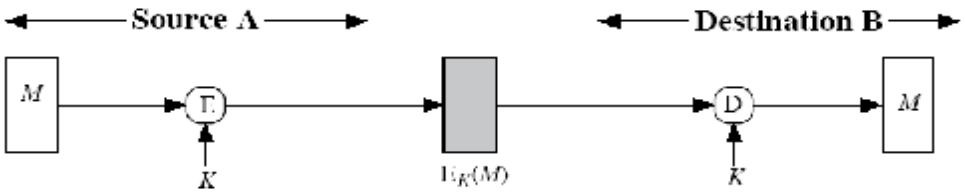
Message encryption – the cipher text of the entire message serves as its authenticator.

Message authentication code (MAC) – a public function of the message and a secret key that produces a fixed length value serves as the authenticator.

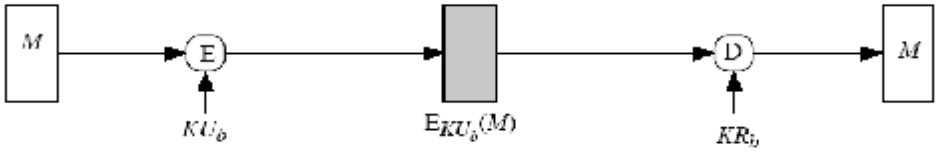
Hash function – a public function that maps a message of any length into a fixed length hash value, which serves as the authenticator.

Message encryption

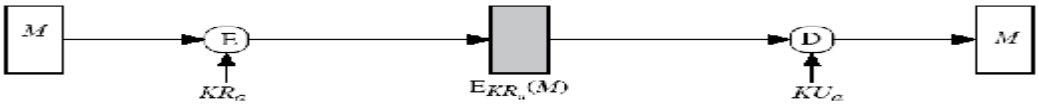
Message encryption by itself can provide a measure of authentication. The analysis differs from symmetric and public key encryption schemes.



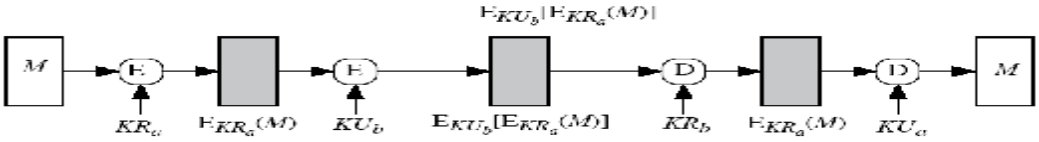
(a) Symmetric encryption: confidentiality and authentication



(b) Public key encryption: confidentiality



(c) Public-key encryption: authentication and signature

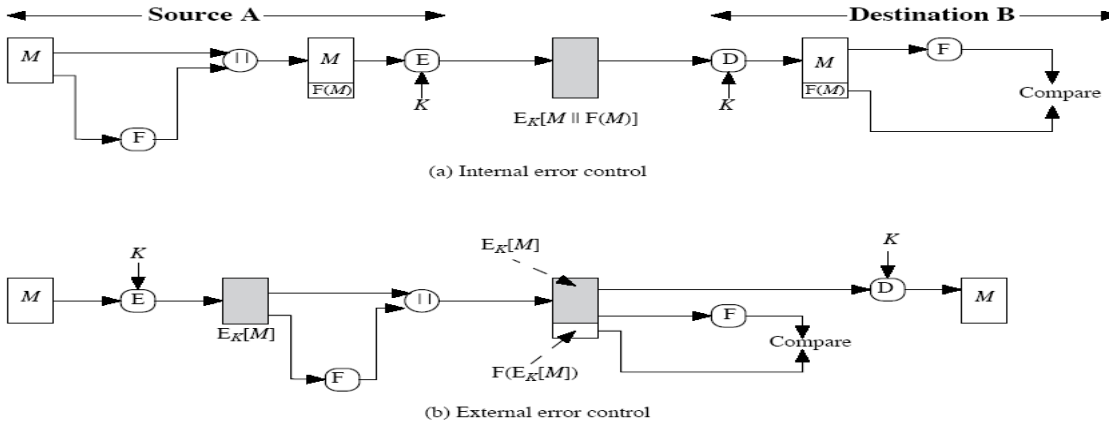


(d) Public-key encryption: confidentiality, authentication, and signature

Suppose the message can be any arbitrary bit pattern. In that case, there is no way to determine automatically, at the destination whether an incoming message is the ciphertext of a legitimate message. One solution to this problem is to force the plaintext to have some structure that is easily recognized but that cannot be replicated without recourse to the encryption function. We could, for example, append an error detecting code, also known as Frame Check Sequence (FCS) or checksum to each message before encryption

'A' prepares a plaintext message M and then provides this as input to a function F that produces an FCS. The FCS is appended to M and the entire block is then encrypted. At the destination, B decrypts the incoming block and treats the result as a message with an appended FCS. B applies the same function F to attempt to reproduce the FCS. If the calculated FCS is equal to the incoming FCS, then the message is considered authentic.

In the internal error control, the function F is applied to the plaintext, whereas in external error control, F is applied to the ciphertext (encrypted message).



(a) Internal error control

(b) External error control

MESSAGE AUTHENTICATION CODE (MAC)

An alternative authentication technique involves the use of secret key to generate a small fixed size block of data, known as cryptographic checksum or MAC that is appended to the message. This technique assumes that two communication parties say A and B, share a common secret key 'k'. When A has to send a message to B, it calculates the MAC as a function of the message and the key.

MAC = $C_K(M)$ Where M – input message

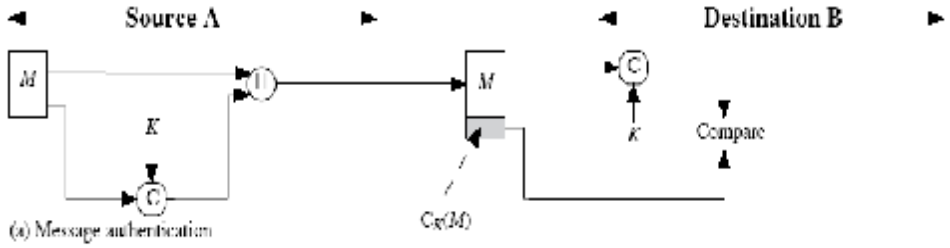
C – MAC function

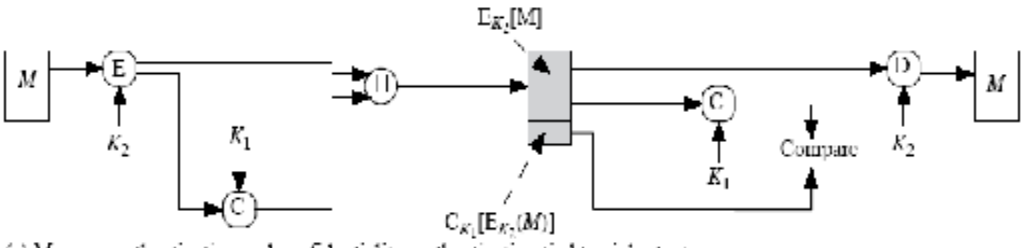
K – Shared secret key

+MAC - Message Authentication Code

The message plus MAC are transmitted to the intended recipient. The recipient performs the same calculation on the received message, using the shared secret key, to generate a new MAC. The received MAC is compared to the calculated MAC. If it is equal, then the message is considered authentic.

A MAC function is similar to encryption. One difference is that MAC algorithm need not be reversible, as it must for decryption. In general, the MAC function is a many- to-one function.





Requirements for MAC:

When an entire message is encrypted for confidentiality, using either symmetric or asymmetric encryption, the security of the scheme generally depends on the bit length of the key. Barring some weakness in the algorithm, the opponent must resort to a brute- force attack using all possible keys. On average, such an attack will require $2^{(k-1)}$ attempts for a k-bit key.

In the case of a MAC, the considerations are entirely different. Using brute-force methods, how would an opponent attempt to discover a key?

If confidentiality is not employed, the opponent has access to plaintext messages and their associated MACs. Suppose k > n; that is, suppose that the key size is greater than the MAC size. Then, given a known M_1 and MAC_1, with $MAC_1 = CK(M_1)$, the cryptanalyst can perform $MAC_i = CK_i(M_1)$ for all possible key values K_i.

At least one key is guaranteed to produce a match of $MAC_i = MAC_1$.

Note that a total of 2^k MACs will be produced, but there are only $2^n < 2^k$ different MAC values. Thus, a number of keys will produce the correct MAC and the opponent has no way of knowing which is the correct key. On average, a total of $2^k/2^n = 2^{(k-n)}$ keys will produce a match. Thus, the opponent must iterate the attack:

Round 1

Given: M_1, $MAC_1 = CK(M_1)$

Compute $MAC_i = CK_i(M_1)$ for all 2^k keys

Number of matches $\approx 2^{(k-n)}$

Round 2

Given: M_2, $MAC_2 = CK(M_2)$

Compute $MAC_i = CK_i(M_2)$ for the $2^{(k-n)}$ keys resulting from Round 1

Number of matches $\approx 2^{(k-2xn)}$

and so on. On average, a rounds will be needed if k = a x n. For example, if an 80-bit key is used and the MAC is 32 bits long, then the first round will produce about 2^{48} possible keys. The second round will narrow the possible keys to about 2^{16} possibilities. The third round should produce only a single key, which must be the one used by the sender.

If the key length is less than or equal to the MAC length, then it is likely that a first round will produce a single match.

Thus, a brute-force attempt to discover the authentication key is no less effort and may be more effort than that required to discover a decryption key of the same length. However, other attacks that do not require the discovery of the key are possible.

Consider the following MAC algorithm. Let $M = (X_1||X_2||...||X_m)$ be a message that is treated as a concatenation of 64-bit blocks X_i. Then define

$\Delta(M) = X1 \oplus X2 \oplus \ldots \oplus Xm$

$Ck(M) = Ek(\Delta(M))$

where $\oplus$ is the exclusive-OR (XOR) operation and the encryption algorithm is DES in electronic codebook mode. Thus, the key length is 56 bits and the MAC length is 64 bits. If an opponent observes $\{M||C(K, M)\}$, a brute-force attempt to determine K will require at least 2^{56} encryptions. But the opponent can attack the system by replacing X_1 through X_{m-1} with any desired values Y_1 through Y_{m-1} and replacing X_m with Y_m where Y_m is calculated as follows:

$Y_m = Y_1 \oplus Y_2 \oplus \ldots \oplus Y_{m-1} \oplus \Delta(M)$

The opponent can now concatenate the new message, which consists of Y_1 through Y_m, with the original MAC to form a message that will be accepted as authentic by the receiver. With this tactic, any message of length 64 x (m-1) bits can be fraudulently inserted.

Then the MAC function should satisfy the following requirements: The MAC function should have the following properties:

If an opponent observes M and $C_K(M)$, it should be computationally infeasible for the opponent to construct a message M' such that $C_K(M') = C_K(M)$

$C_K(M)$ should be uniformly distributed in the sense that for randomly chosen messages, M and M', the probability that $C_K(M) = C_K(M')$ is 2^{-n} where n is the number of bits in the MAC.

Let M' be equal to some known transformation on M. i.e., M' = f(M).

MAC based on DES

One of the most widely used MACs, referred to as Data Authentication Algorithm (DAA) is based on DES.

The algorithm can be defined as using cipher block chaining (CBC) mode of operation of DES with an initialization vector of zero. The data to be authenticated are grouped into contiguous 64-bit blocks: $D_1, D_2 \ldots D_n$. if necessary, the final block is padded on the right with zeros to form a full 64-bit block. Using the DES encryption algorithm and a secret key, a data authentication code

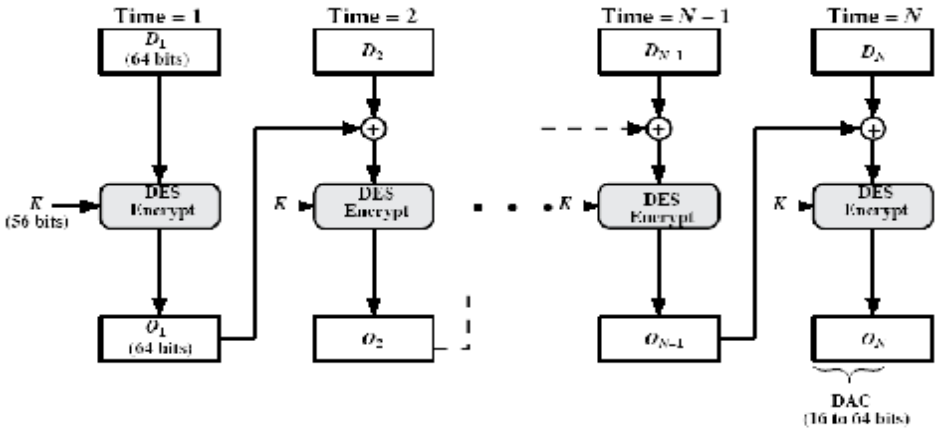
(DAC) is calculated as follows:

$O_1 = E_K(D_1)$

$O_2 = E_K(D_2 \oplus O_1)$

$O_3 = E_K(D_3 \oplus O_2) \ldots$

$O_N = E_K(D_N \oplus O_{N-1})$

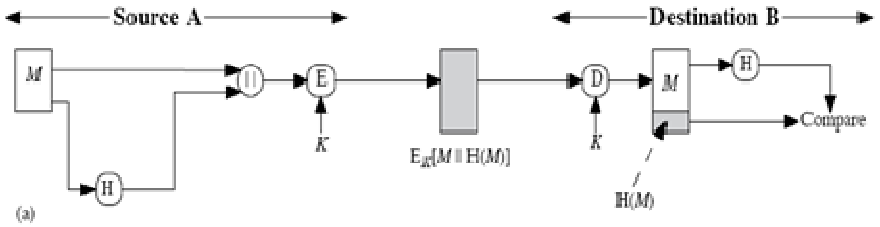


HASH FUNCTIONS

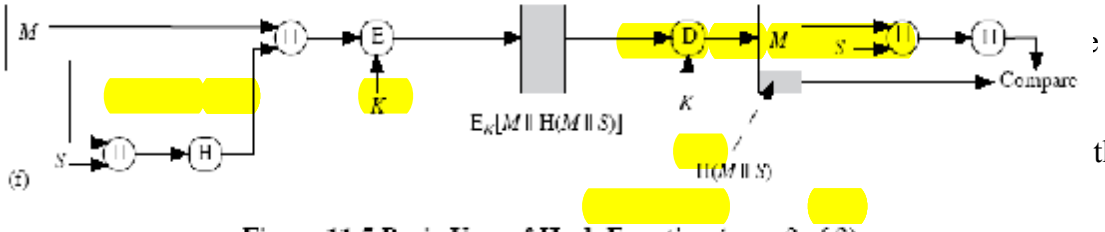
A variation on the message authentication code is the one way hash function. As with MAC, a hash function accepts a variable size message M as input and produces a fixed-size output, referred to as hash code H(M). Unlike a MAC, a hash code does not use a key but is a function only of the input message. The hash code is also referred to as a message digest or hash value.

There are varieties of ways in which a hash code can be used to provide message authentication, as follows:

a) The message plus the hash code is encrypted using symmetric encryption. This is identical to that of internal error control strategy. Because encryption is applied to the entire message plus the hash code, confidentiality is also provided.



b) Only the hash code is encrypted, using symmetric encryption. This reduces the processing burden for those applications that do not require confidentiality.



the public key

Figure 11.5 Basic Uses of Hash Function (page 2 of 2)

e) This technique uses a hash function, but no encryption for message authentication. This technique assumes that the two communicating parties share a common secret value 'S'. The source computes the hash value over the concatenation of M and S and appends the resulting hash value to M.

f) Confidentiality can be added to the previous approach by encrypting the entire message plus the hash code.

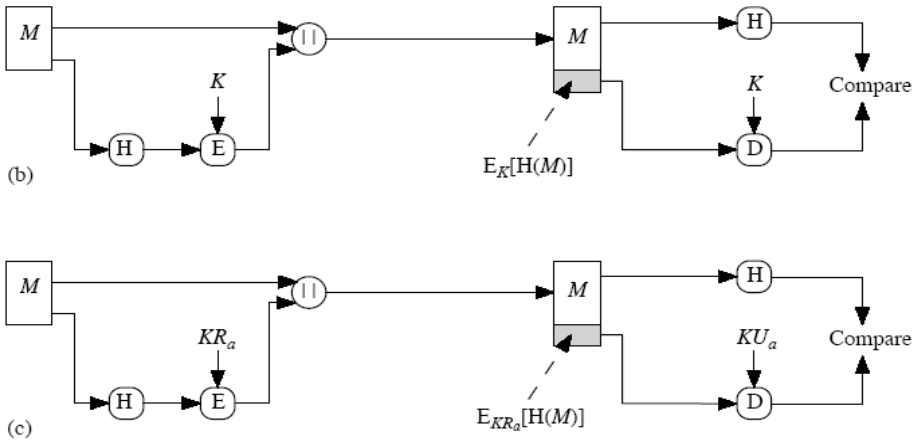


Figure 11.5 Basic Uses of Hash Function (page 1 of 2)

A hash value h is generated by a function H of the form h = H(M)

Where M is a variable-length message and H(M) is the fixed-length hash value. The hash value is appended to the message at the source at a time when the message is assumed or known to be correct. The receiver authenticates that message by re-computing the hash value.

Requirements for a Hash Function

1. H can be applied to a block of data of any size.

2. H produces a fixed-length output.
3. H(x) is relatively easy to compute for any given x, making both hardware and software implementations practical.
4. For any given value h, it is computationally infeasible to find x such that H(x) = h. This is sometimes referred to in the literature as the one-way property.

5. For any given block x, it is computationally infeasible to find y x such that H(y) = H(x). This is sometimes referred to as **weak collision resistance**.
6. It is computationally infeasible to find any pair (x, y) such that H(x) = H(y). This is sometimes referred to as **strong collision resistance**.

The first three properties are requirements for the practical application of a hash function to message authentication. The fourth property, the one-way property, states that it is easy to generate a code given a message but virtually impossible to generate a message given a code. The fifth property guarantees that an alternative message hashing to the same value as a given message cannot be found. This prevents forgery when an encrypted hash code is used. The sixth property refers to how resistant the hash function is to a type of attack known as the birthday attack, which we examine shortly.

Simple Hash Functions

All hash functions operate using the following general principles. The input (message, file, etc.) is viewed as a sequence of n-bit blocks. The input is processed one block at a time in an iterative fashion to produce an n-bit hash function.

One of the simplest hash functions is the bit-by-bit exclusive-OR (XOR) of every block. This can be expressed as follows:

$C_i = b_{i1} \oplus b_{i1} \oplus \ldots \oplus b_{im}$

Where

C_i = ith bit of the hash code, $1 \leq i \leq n$

m = number of n-bit blocks in the input b_{ij} = ith bit in jth block

$\oplus$ = XOR operation

Thus, the probability that a data error will result in an unchanged hash value is 2^n. With more

predictably formatted data, the function is less effective. For example, in most normal text files, the high-order bit of each octet is always zero. So if a 128-bit hash value is used, instead of an effectiveness of 2^{128}, the hash function on this type of data has an effectiveness of 2^{112}.

A simple way to improve matters is to perform a one-bit circular shift, or rotation, on the hash value after each block is processed. The procedure can be summarized as follows:

1. Initially set the n-bit hash value to zero.
2. Process each successive n-bit block of data as follows:

a. Rotate the current hash value to the left by one bit. b. XOR the block into the hash value.

Birthday Attacks

Suppose that a 64-bit hash code is used. One might think that this is quite secure. For example, if an encrypted hash code C is transmitted with the corresponding unencrypted

Message M, then an opponent would need to find an M' such that H(M') = H(M) to substitute another message and fool the receiver.

On average, the opponent would have to try about 2^{63} messages to find one that matches the hash code of the intercepted message

However, a different sort of attack is possible, based on **the birthday paradox** The source, A, is prepared to "sign" a message by appending the appropriate m-bit hash code and encrypting that hash code with A's private key

1. The opponent generates $2^{m/2 \text{ variations}}$ on the message, all of which convey essentially the same meaning. (Fraudulent message
2. The two sets of messages are compared to find a pair of messages that produces the same hash code. The probability of success, by the birthday paradox, is greater than 0.5. If no match is found, additional valid and fraudulent messages are generated until a match is made.
3. The opponent offers the valid variation to A for signature. This signature can then be attached to the fraudulent variation for transmission to the intended recipient. Because the two variations have the same hash code, they will produce the same signature; the opponent is assured of success even though the encryption key is not known.

Thus, if a 64-bit hash code is used, the level of effort required is only on the order of 2^{32} .

Block Chaining Techniques

Divide a message M into fixed-size blocks $M_1, M_2, ..., M_N$ and use a symmetric encryption system

such as DES to compute the hash code G as follows:

H_0 = initial value

$H_i = E_{Mi}[H_{i-1}] \; G = H_N$

This is similar to the CBC technique, but in this case there is no secret key. As with any hash code, this scheme is subject to the birthday attack, and if the encryption algorithm is DES and only a 64-bit hash code is produced, then the system is vulnerable.

Furthermore, another version of the birthday attack can be used even if the opponent has access to only one message and its valid signature and cannot obtain multiple signings.

Here is the scenario; we assume that the opponent intercepts a message with a signature in the form of an encrypted hash code and that the unencrypted hash code is m bits long:

1. Use the algorithm defined at the beginning of this subsection to calculate the unencrypted hash code G.
2. Construct any desired message in the form $Q_1, Q_2, ..., Q_{N2}$.
3. Compute for $H_i = EQ_i [H_{i-1}]$ for $1 \leq i \leq (N-2)$.
4. Generate $2^{m/2}$ random blocks; for each block X, compute $E_X[H_{N-2}.]$ Generate an additional $2^{m/2}$ random blocks; for each block Y, compute $D_Y[G]$, where D is the decryption function corresponding to E.
5. Based on the birthday paradox, with high probability there will be an X and Y such that $E_X [H_{N-2}] = D_Y[G]$.
6. Form the message $Q_1, Q_2, ..., Q_{N-2}$, X, Y. This message has the hash code G and therefore can be used with the intercepted encrypted signature.

This form of attack is known as a **meet-in-the-middle attack**.

Security of Hash Functions and Macs

Just as with symmetric and public-key encryption, we can group attacks on hash functions and MACs into two categories: brute-force attacks and cryptanalysis.

Brute-Force Attacks

The nature of brute-force attacks differs somewhat for hash functions and MACs.

Hash Functions

The strength of a hash function against brute-force attacks depends solely on the length of the hash

code produced by the algorithm. Recall from our discussion of hash functions that there are three desirable properties:

One-way: For any given code h, it is computationally infeasible to find x such that H(x) = h.

Weak collision resistance: For any given block x, it is computationally infeasible to find y x with H(y) = H(x).

Strong collision resistance: It is computationally infeasible to find any pair (x, y) Such that H(x) = H(y).

For a hash code of length n, the level of effort required, as we have seen is proportional to the following:

One way	2^n
Weak collision resistance	2^n
Strong collision resistance	$2^{n/2}$

Cryptanalysis

As with encryption algorithms, cryptanalytic attacks on hash functions and MAC algorithms seek to exploit some property of the algorithm to perform some attack other than an exhaustive search.

Hash Functions

In recent years, there has been considerable effort, and some successes, in developing cryptanalytic attacks on hash functions. To understand these, we need to look at the overall structure of a typical secure hash function, and is the structure of most hash functions in use today, including SHA and Whirlpool.

The hash function takes an input message and partitions it into L fixed-sized blocks of b bits each. If necessary, the final block is padded to b bits.

The final block also includes the value of the total length of the input to the hash function.The inclusion of the length makes the job of the opponent more difficult.

Either the opponent must find two messages of equal length that hash to the same value or two messages of differing lengths that, together with their length values, hash to the same value.

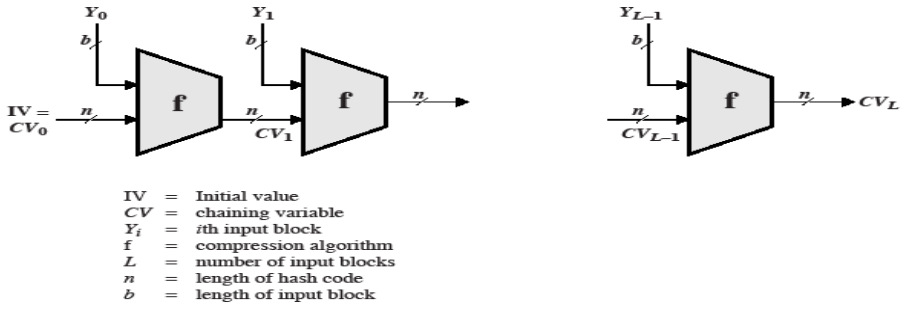


Figure 11.10 General Structure of Secure Hash Code

The hash algorithm involves repeated use of a **compression function**, f, that takes two inputs (an n-bit input from the previous step, called the chaining variable, and a b-bit block) and produces an n-bit output. At the start of hashing, the chaining variable has an initial value that is specified as part of the algorithm. The final value of the chaining variable is the hash value. Often, b > n; hence the term compression. The hash function can be summarized as follows:

$$CV_0 = IV = \text{initial n-bit value } CV_i = f(CV_{i-1}, Y_{i-1})\ 1 \le i \le L\ H(M) = CV_L$$

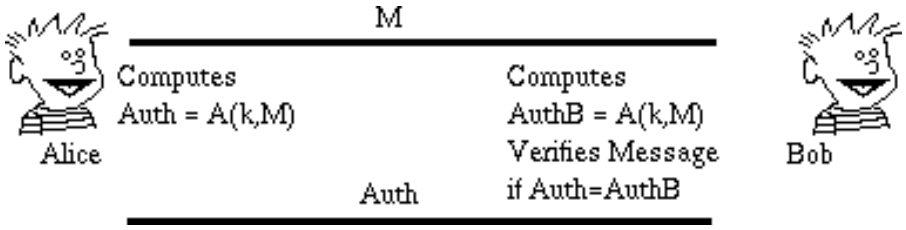
Where the input to the hash function is a message M consisting of the blocks $Y_0, Y_1, ..., Y_{L-1}$. The structure can be used to produce a secure hash function to operate on a message of any length.

Message Authentication Codes

There is much more variety in the structure of MACs than in hash functions, so it is difficult to generalize about the cryptanalysis of MACs. Further, far less work has been done on developing such attacks.

Message Authentication.

- Message authentication is concerned with:
  - protecting the integrity of a message
  - validating identity of originator
  - non-repudiation of origin (dispute resolution)
- electronic equivalent of a signature on a message
- An **authenticator**, **signature**, or **message authentication code (MAC)** is sent along with the message
- The MAC is generated via some algorithm which depends on both the message and some (public or private) key known only to the sender and receiver
- The message may be of any length
- the MAC may be of any length, but more often is some fixed size, requiring the use of some **hash function** to condense the message to the required size if this is not acheived by the authentication scheme
- need to consider replay problems with message and MAC
  - require a message sequence number, timestamp or negotiated random values



Authentication using Private-key Ciphers

- if a message is being encrypted using a session key known only to the sender and receiver, then the message may also be authenticated
  - since only sender or receiver could have created it
  - any interference will corrupt the message (provided it includes sufficient redundancy to detect change)

- but this does not provide non-repudiation since it is impossible to prove who created the message

- message authentication may also be done using the standard modes of use of a block cipher

- sometimes do not want to send encrypted messages

- can use either CBC or CFB modes and send final block, since this will depend on all previous bits of the message

- no hash function is required, since this method accepts arbitrary length input and produces a fixed output

- usually use a fixed known IV

- this is the approached used in Australian EFT standards AS8205

- major disadvantage is small size of resulting MAC since 64-bits is probably too small

Hashing Functions

- hashing functions are used to condense an arbitrary length message to a fixed size, usually for subsequent signature by a digital signature algorithm

- good cryptographic hash function h should have the following properties:

- h should destroy all holomorphic structures in the underlying public key cryptosystem (be unable to compute hash value of 2 messages combined given their individual hash values)

- h should be computed on the entire message

- h should be a one-way function so that messages are not disclosed by their signatures

- it should be computationally infeasible given a message and its hash value to compute another message with the same hash value

- should resist **birthday attacks** (finding any 2 messages with the same hash value, perhaps by iterating through minor permutations of 2 messages)

- it is usually assumed that the hash function is public and not keyed

- traditional CRCs do not satisfy the above requirements

- length should be large enough to resist birthday attacks (64-bits is now regarded as too small, 128-512 proposed)

MD2, MD4 and MD5

- family of one-way hash functions by Ronald Rivest
- MD2 is the oldest, produces a 128-bit hash value, and is regarded as slower and less secure than MD4 and MD5
- MD4 produces a 128-bit hash of the message, using bit operations on 32-bit operands for fast implementation

R L Rivest, "The MD4 Message Digest Algorithm", Advances in Cryptology - Crypto'90, Lecture Notes in Computer Science No 537, Springer-Verlag 1991, pp303-311

- MD4 overview
  - pad message so its length is 448 mod 512
  - append a 64-bit message length value to message
  - initialise the 4-word (128-bit) buffer (A,B,C,D)
  - process the message in 16-word (512-bit) chunks, using 3 rounds of 16 bit operations each on the chunk & buffer
  - output hash value is the final buffer value
- some progress at cryptanalysing MD4 has been made, with a small number of collisions having been found
- MD5 was designed as a strengthened version, using four rounds, a little more complex than in MD4 [2]
- a little progress at cryptanalysing MD5 has been made with a small number of collisions having been found
- both MD4 and MD5 are still in use and considered secure in most practical applications
- both are specified as Internet standards (MD4 in RFC1320, MD5 in RFC1321)

3.3.1 SHA (Secure Hash Algorithm)

- SHA was designed by NIST & NSA and is the US federal standard for use with the DSA signature scheme (nb the algorithm is SHA, the standard is SHS)
- it produces 160-bit hash values
- SHA overview[3]
  - pad message so its length is a multiple of 512 bits
  - initialise the 5-word (160-bit) buffer (A,B,C,D,E) to
  - (67452301,efcdab89,98badcfe,10325476,c3d2e1f0)
  - process the message in 16-word (512-bit) chunks, using 4 rounds of 20 bit operations each on the chunk & buffer
  - output hash value is the final buffer value
- SHA is a close relative of MD5, sharing much common design, but each having differences
- SHA has very recently been subject to modification following NIST identification of some concerns, the exact nature of which is not public
- current version is regarded as secure

Digital Signature Schemes

- public key signature schemes
- the private-key signs (creates) signatures, and the public-key verifies signatures
- only the owner (of the private-key) can create the digital signature, hence it can be used to verify who created a message
- anyone knowing the public key can verify the signature (provided they are confident of the identity of the owner of the public key - the key distribution problem)
- usually don't sign the whole message (doubling the size of information exchanged), but just a **hash** of the message

- digital signatures can provide non-repudiation of message origin, since an asymmetric algorithm is used in their creation, provided suitable timestamps and redundancies are incorporated in the signature

RSA

- RSA encryption and decryption are commutative, hence it may be used directly as a digital signature scheme
  - given an RSA scheme {(e,R), (d,p,q)}
- to **sign** a message, compute:
  - $S = M^d (\text{mod } R)$
- to **verify** a signature, compute:
  - $M = S^e (\text{mod } R) = M^{e.d} (\text{mod } R) = M (\text{mod } R)$
- thus know the message was signed by the owner of the public-key
- would seem obvious that a message may be encrypted, then signed using RSA without increasing it size
  - but have blocking problem, since it is encrypted using the receivers modulus, but signed using the senders modulus (which may be smaller)
  - several approaches possible to overcome this
- more commonly use a hash function to create a separate MDC which is then signed

El Gamal Signature Scheme

- whilst the ElGamal encryption algorithm is not commutative, a closely related signature scheme exists
- El Gamal Signature scheme
- given prime p, public random number g, private (key) random number x, compute
  - $y = g^x (\text{mod } p)$
- public key is (y,g,p)

- nb (g,p) may be shared by many users
- p must be large enough so discrete log is hard

- private key is (x)
- to **sign** a message M
  - choose a random number k, GCD(k,p-1)=1
  - compute $a = g^k \pmod p$
  - use extended Euclidean (inverse) algorithm to solve
  - $M = x.a + k.b \pmod{p-1}$
  - the signature is (a,b), k must be kept secret
  - (like ElGamal encryption is double the message size)
- to **verify** a signature (a,b) confirm:
  - $y^a.a^b \pmod p = g^M \pmod p$

Example of ElGamal Signature Scheme

- given p=11, g=2
- choose private key x=8
- compute
  - $y = g^x \pmod p = 2^8 \pmod{11} = 3$
- public key is y=3,g=2,p=11)
- to sign a message M=5
  - choose random k=9
  - confirm gcd(10,9)=1
  - compute
    - $a = g^k \pmod p = 2^9 \pmod{11} = 6$

- solve
  - $M = x.a+k.b(\text{mod } p-1)$
  - $5 = 8.6+9.b(\text{mod } 10)$
  - giving $b = 3$
- signature is (a=6,b=3)

- to verify the signature, confirm the following are correct:
  - $y^{a}.a^{b}(\text{mod } p) = g^{M}(\text{mod } p)$
  - $3^{6}.6^{3}(\text{mod } 11) = 2^{5}(\text{mod } 11)$

DSA (Digital Signature Algorithm)

- DSA was designed by NIST & NSA and is the US federal standard signature scheme (used with SHA hash alg)
  - DSA is the algorithm, DSS is the standard
  - There was considerable reaction to its announcement!
    - debate over whether RSA should have been used
    - debate over the provision of a signature only alg
- DSA is a variant on the ElGamal and Schnorr algorithms
- description of DSA
  - $p = 2^{L}$ a prime number, where L= 512 to 1024 bits and is a multiple of 64
  - q a 160 bit prime factor of p-1
  - $g = h^{(p-1)/q}$ where h is any number less than p-1 with $h^{(p-1)/q}(\text{mod } p) > 1$
  - x a number less than q
  - $y = g^{x}(\text{mod } p)$
- to **sign** a message M

- generate random k, k<q
- compute
  - $r = (g^k \pmod p) \pmod q$
  - $s = k^{-1}.SHA(M) + x.r \pmod q$
- the signature is (r,s)

- to **verify** a signature:
  - $w = s^{-1} \pmod q$
  - $u1 = (SHA(M).w) \pmod q$
  - $u2 = r.w \pmod q$
  - $v = (gu1.yu2 \pmod p) \pmod q$
  - if v=r then the signature is verified

- comments on DSA
  - was originally a suggestion to use a common modulus, this would make a tempting target, discouraged
  - it is possible to do both ElGamal and RSA encryption using DSA routines, this was probably not intended :-)
  - DSA is patented with royalty free use, but this patent has been contested, situation unclear
  - Gus Simmons has found a subliminal channel in DSA, could be used to leak the private key from a library - make sure you trust your library implementer

MODULE - IV

AUTHENTICATION SERVICES KERBEROS

Kerberos provides a centralized authentication server whose function is to authenticate users to servers and servers to users. Kerberos relies exclusively on conventional encryption, making no use of public-key encryption.

The following are the requirements for Kerberos:

- **Secure:** A network eavesdropper should not be able to obtain the necessary information to impersonate a user. More generally, Kerberos should be strong enough that a potential opponent does not find it to be the weak link.
- **Reliable:** For all services that rely on Kerberos for access control, lack of availability of the Kerberos service means lack of availability of the supported services. Hence, Kerberos should be highly reliable and should employ a distributed server architecture, with one system able to back up another.
- **Transparent:** Ideally, the user should not be aware that authentication is taking place, beyond the requirement to enter a password.
- **Scalable:** The system should be capable of supporting large numbers of clients and servers. This suggests a modular, distributed architecture.

To support these requirements, the overall scheme of Kerberos is that of a trusted third- party authentication service that uses a protocol based on that proposed by Needham and Schroeder [NEED78] It is trusted in the sense that clients and servers trust Kerberos to mediate their mutual authentication. Assuming the Kerberos protocol is well designed, then the authentication service is secure if the Kerberos server itself is secure.

A simple authentication dialogue

In an unprotected network environment, any client can apply to any server for service. The obvious security risk is that of impersonation. To counter this threat, servers must be able to confirm the identities of clients who request service. But in an open environment, this places a substantial burden on each server.

An alternative is to use an authentication server (AS) that knows the passwords of all users and stores these in a centralized database. In addition, the AS shares a unique secret key with each server. The simple authentication dialogue is as follows:

A more secure authentication dialogue

There are two major problems associated with the previous approach:

Plaintext transmission of the password.

Each time a user has to enter the password.

To solve these problems, we introduce a scheme for avoiding plaintext passwords, and anew server, known as ticket granting server (TGS). The hypothetical scenario is as follows:

Once per user logon session:

1. C >> AS: IDc||IDtgs
2. AS >> C: Ek_c ($Ticket_{tgs}$)

Once per type of service:

3. C >> TGS: IDc||IDv||$Ticket_{tgs}$
4. TGS >> C: $ticket_V$

Once per service session:

5. C >> V: IDc||$ticket_V$

$Ticket_{tgs}$= Ek_{tgs}(IDc||ADc||IDtgs||TS1||Lifetime1) $Ticket_V$= Ek_V(IDc||ADc||IDv||TS2||Lifetime2)

C: Client, AS: Authentication Server, V: Server, IDc : ID of the client, Pc:Password of the client, ADc: Address of client, IDv : ID of the server, Kv: secret key shared by AS and V, ||: concatenation, IDtgs: ID of the TGS server, TS1, TS2: time stamps, lifetime: lifetime of the ticket.

The new service, TGS, issues tickets to users who have been authenticated to AS. Thus, the user first requests a ticket-granting ticket ($Ticket_{tgs}$) from the AS. The client module in the user workstation saves this ticket. Each time the user requires access to a new service, the client applies to the TGS, using the ticket to authenticate itself. The TGS then grants a ticket for the particular service. The client saves each service-granting ticket and uses it to authenticate its user to a server each time a particular service is requested. Let us look at the details of this scheme:

1. The client requests a ticket-granting ticket on behalf of the user by sending its user's ID and password to the AS, together with the TGS ID, indicating a request to use the TGS service.

2. The AS responds with a ticket that is encrypted with a key that is derived from the user's password.

When this response arrives at the client, the client prompts the user for his or her password, generates the key, and attempts to decrypt the incoming message.

If the correct password is supplied, the ticket is successfully recovered.Because only the correct user should know the password, only the correct user can recover the ticket. Thus, we have used the password to obtain credentials from Kerberos without having to transmit the password in plaintext.

Now that the client has a ticket-granting ticket, access to any server can be obtained with steps 3 and 4:

3. The client requests a service-granting ticket on behalf of the user. For this purpose, the client transmits a message to the TGS containing the user's ID, the ID of the desired service, and the ticket-granting ticket.

4. The TGS decrypts the incoming ticket and verifies the success of the decryption by the presence of its ID. It checks to make sure that the lifetime has not expired. Then it compares the user ID and network address with the incoming information to authenticate the user. If the user is permitted access to the server V, the TGS issues a ticket to grant access to the requested service.

The service-granting ticket has the same structure as the ticket-granting ticket. Indeed, because the TGS is a server, we would expect that the same elements are needed to authenticate a client to the TGS and to authenticate a client to an application server. Again, the ticket contains a timestamp and lifetime. If the user wants access to the same service at a later time, the client can simply use the previously acquired service-granting ticket and need not bother the user for a password. Note that the ticket is encrypted with a secret key (K_V) known only to the TGS and the server, preventing alteration.

Finally, with a particular service-granting ticket, the client can gain access to the corresponding service with step 5:

5. The client requests access to a service on behalf of the user. For this purpose, the client transmits a message to the server containing the user's ID nd the service-granting ticket. The server authenticates by using the contents of the ticket.

This new scenario satisfies the two requirements of only one password query per user session and

protection of the user password.

Kerbero V4 Authentication Dialogue Message Exchange

Two additional problems remain in the more secure authentication dialogue:

Lifetime associated with the ticket granting ticket. If the lifetime is very short, then the user will be repeatedly asked for a password. If the lifetime is long, then the opponent has the greater opportunity for replay.

Requirement for the servers to authenticate themselves to users. The actual Kerberos protocol version 4 is as follows:

- a basic third-party authentication scheme
- have an Authentication Server (AS)

– users initially negotiate with AS to identify self

–AS provides a non-corruptible authentication credential (ticket granting ticket TGT)

- have a Ticket Granting server (TGS)

–users subsequently request access to other services from TGS on basis of users

Message (1)	**Client requests ticket-granting ticket**
ID_C	Tells AS identity of user from this client
ID_{tgs}	Tells AS that user requests access to TGS
TS_1	Allows AS to verify that client's clock is synchronized with that of AS
Message (2)	**AS returns ticket-granting ticket**
K_c	Encryption is based on user's password, enabling AS and client to verify asswo d, and p t ctin contents of m ssa e (2)
$K_{c,tgs}$	Copy of session key accessible to client created by AS to permit secure exchange between client and TGS without requiring them to share a
ID_{tgs}	Confirms that this ticket is for the TGS

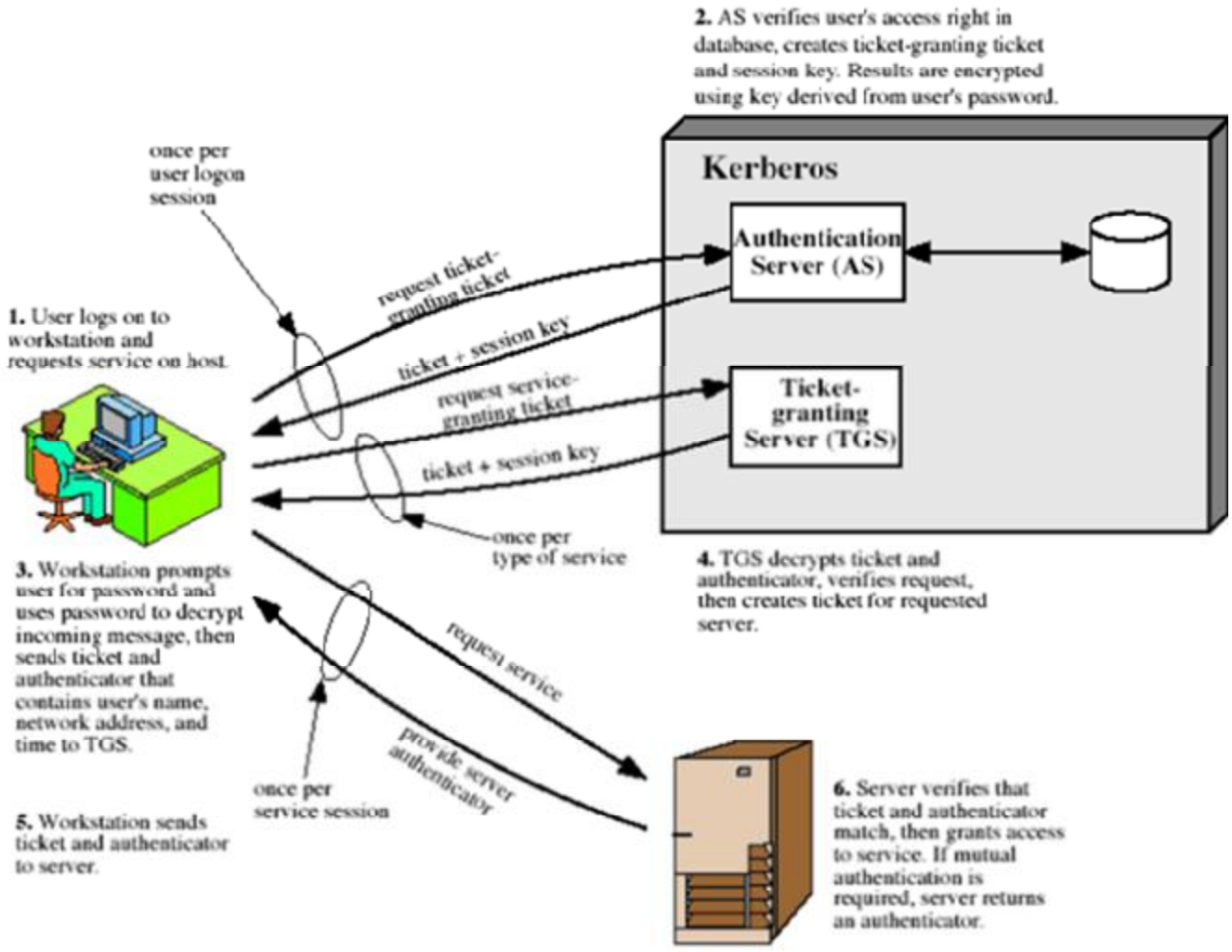
The table given below illustrates the mode of dialogue in V4

(a) Authentication Service Exchange: to obtain ticket-granting ticket
(1) C → AS: $ID_c \parallel ID_{tgs} \parallel TS_1$
(2) AS → C: $\mathrm{E}_{K_c}\left[K_{c,tgs} \parallel ID_{tgs} \parallel TS_2 \parallel Lifetime_2 \parallel Ticket_{tgs}\right]$
$Ticket_{tgs} = \mathrm{E}_{K_{tgs}}\left[K_{c,tgs} \parallel ID_C \parallel AD_C \parallel ID_{tgs} \parallel TS_2 \parallel Lifetime_2\right]$
(b) Ticket-Granting Service Exchange: to obtain service-granting ticket
(3) C → TGS: $ID_v \parallel Ticket_{tgs} \parallel Authenticator_c$
(4) TGS → C: $\mathrm{E}_{K_{c,tgs}}\left[K_{c,v} \parallel ID_v \parallel TS_4 \parallel Ticket_v\right]$
$Ticket_{tgs} = \mathrm{E}_{K_{tgs}}\left[K_{c,tgs} \parallel ID_C \parallel AD_C \parallel ID_{tgs} \parallel TS_2 \parallel Lifetime_2\right]$
$Ticket_v = \mathrm{E}_{K_v}\left[K_{c,v} \parallel ID_C \parallel AD_C \parallel ID_v \parallel TS_4 \parallel Lifetime_4\right]$
$Authenticator_c = \mathrm{E}_{K_{tgs}}\left[ID_C \parallel AD_C \parallel TS_3\right]$
(c) Client/Server Authentication Exchange: to obtain service
(5) C → V: $Ticket_v \parallel Authenticator_c$
(6) V → C: $\mathrm{E}_{K_{c,v}}\left[TS_5 + 1\right]$ (for mutual authentication)
$Ticket_v = \mathrm{E}_{K_v}\left[K_{c,v} \parallel ID_C \parallel AD_C \parallel ID_v \parallel TS_4 \parallel Lifetime_4\right]$
$Authenticator_c = \mathrm{E}_{K_{c,v}}\left[ID_C \parallel AD_C \parallel TS_5\right]$

TS_2	Informs client of time this ticket was issued
$Lifetime_2$	Informs client of the lifetime of this ticket
$Ticket_{tgs}$	Ticket to be used by client to access TGS
	(a) Authentication Service Exchange
Message (3)	**Client requests service-granting ticket**
ID_V	Tells TGS that user requests access to server V
$Ticket_{tgs}$	Assures TGS that this user has been authenticated by AS
$Authenticator_c$	Generated by client to validate ticket
Message (4)	**TGS returns service-granting ticket**
$K_{c,tgs}$	Key shared only by C and TGS protects contents of message (4)
$K_{c,v}$	Copy of session key accessible to client created by TGS to permit secure exchange between client and server without requiring them to share a
ID_V	Confirms that this ticket is for server V
TS_4	Informs client of time this ticket was issued
$Ticket_V$	Ticket to be used by client to access server V
$Ticket_{tgs}$	Reusable so that user does not have to reenter password
K_{tgs}	Ticket is encrypted with key known only to AS and TGS, to prevent tampering
$K_{c,tgs}$	Copy of session key accessible to TGS used to decrypt authenticator, thereby authenticating ticket
ID_C	Indicates the rightful owner of this ticket
AD_C	Prevents use of ticket from workstation other than one that initially requested the ticket
ID_{tgs}	Assures server that it has decrypted ticket properly
TS_2	Informs TGS of time this ticket was issued
$Lifetime_2$	Prevents replay after ticket has expired
$Authenticator_c$	Assures TGS that the ticket presenter is the same as the client for whom the ticket was issued has very short lifetime to prevent replay
$K_{c,tgs}$	Authenticator is encrypted with key known only to client and TGS, to prevent tampering

ID_C	Must match ID in ticket to authenticate ticket
AD_C	Must match address in ticket to authenticate ticket
TS_3	Informs TGS of time this authenticator was generated
	(b) Ticket-Granting Service Exchange
Message (5)	**Client requests service**
$Ticket_V$	Assures server that this user has been authenticated by AS
$Authenticator_C$	Generated by client to validate ticket
Message (6)	Optional authentication of server to client
$K_{C,V}$	Assures C that this message is from V
$TS_5 + 1$	Assures C that this is not a replay of an old reply
$Ticket_V$	Reusable so that client does not need to request a new ticket from TGS for each access to the same server
K_V	Ticket is encrypted with key known only to TGS and server, to prevent tampering
$K_{C,V}$	Copy of session key accessible to client; used to decrypt authenticator, thereby authenticating ticket
ID_C	Indicates the rightful owner of this ticket
AD_C	Prevents use of ticket from workstation other than one that initially requested the ticket
ID_V	Assures server that it has decrypted ticket properly
TS_4	Informs server of time this ticket was issued
$Lifetime_4$	Prevents replay after ticket has expired
$Authenticator_C$	Assures server that the ticket presenter is the same as the client for whom the ticket was issued; has very short lifetime to prevent replay
$K_{C,V}$	Authenticator is encrypted with key known only to client and server, to prevent tampering
ID_C	Must match ID in ticket to authenticate ticket
AD_C	Must match address in ticket to authenticate ticket
TS_5	Informs server of time this authenticator was generated
	(c) Client/Server Authentication

Kerberos 4 Overview



Kerberos Realms and Multiple Kerberi

A full-service Kerberos environment consisting of a Kerberos server, a number of clients, and a number of application servers requires the following:

1. The Kerberos server must have the user ID and hashed passwords of all participating users in its database. All users are registered with the Kerberos server.
2. The Kerberos server must share a secret key with each server. All servers are registered with the Kerberos server.Such an environment is referred to as a **Kerberos realm**.

The concept of *realm* can be explained as follows.

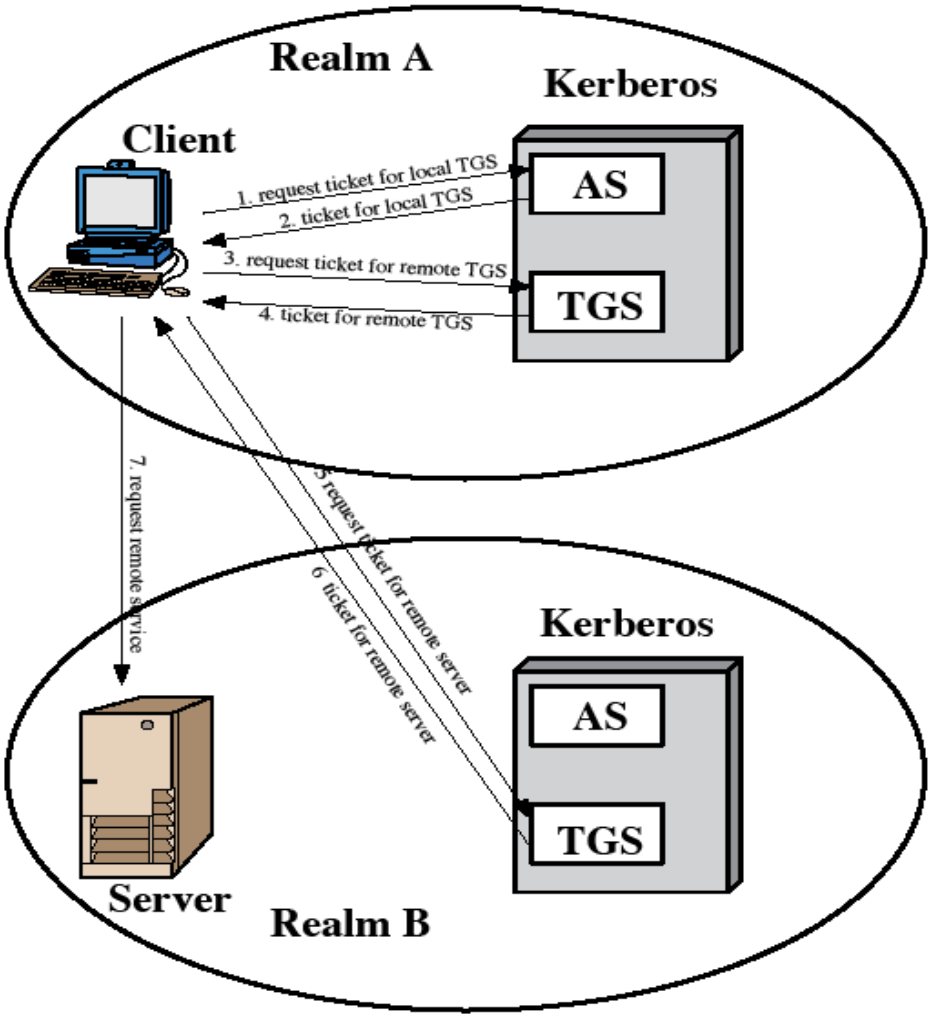


Figure 14.2 Request for Service in Another Realm

A Kerberos realm is a set of managed nodes that share the same Kerberos database. The Kerberos

database resides on the Kerberos master computer system, which should be kept in a physically secure room.

A read-only copy of the Kerberos database might also reside on other Kerberos computer systems. However, all changes to the database must be made on the master computer system. Changing or accessing the contents of a Kerberos database requires the Kerberos master password.

A related concept is that of a Kerberos principal, which is a service or user that is known to the Kerberos system.

Each Kerberos principal is identified by its principal name. Principal names consist of three parts: a service or user name, an instance name, and a realm name

Networks of clients and servers under different administrative organizations typically constitute different realms.

That is, it generally is not practical, or does not conform to administrative policy, to have users and servers in one administrative domain registered with a Kerberos server elsewhere.

However, users in one realm may need access to servers in other realms, and some servers may be willing to provide service to users from other realms, provided that those users are authenticated.

Kerberos provides a mechanism for supporting such interrealm authentication. For two realms to support interrealm authentication, a third requirement is added:

3. The Kerberos server in each interoperating realm shares a secret key with the server in the other realm. The two Kerberos servers are registered with each other.

The scheme requires that the Kerberos server in one realm trust the Kerberos server in the other realm to authenticate its users. Furthermore, the participating servers in the second realm must also be willing to trust the Kerberos server in the first realm.

Kerberos version 5

Version 5 of Kerberos provides a number of improvements over version 4.

- developed in mid 1990's
- provides improvements over v4

– addresses environmental shortcomings and technical deficiencies

- specified as Internet standard RFC 1510

Differences between version 4 and 5

Version 5 is intended to address the limitations of version 4 in two areas:

Environmental shortcomings

- o encryption system dependence
- o internet protocol dependence
- o message byte ordering
- o ticket lifetime
- o authentication forwarding
- o inter-realm authenticaiton

Technical deficiencies

- o double encryption o PCBC encryption o Session keys
- o Password attacks

The version 5 authentication dialogue

(a) Authentication Service Exchange: to obtain ticket-granting ticket
(1) C → AS: Options ‖ ID_c ‖ $Realm_c$ ‖ ID_{tgs} ‖ $Times$ ‖ $Nonce_1$
(2) AS → C: $Realm_c \| ID_C \| Ticket_{tgs} \| E_{K_c}\left[K_{c,tgs} \| Times \| Nonce_1 \| Realm_{tgs} \| ID_{tgs}\right]$
$Ticket_{tgs} = E_{K_{tgs}}\left[Flags \| K_{c,tgs} \| Realm_c \| ID_C \| AD_C \| Times\right]$
(b) Ticket-Granting Service Exchange: to obtain service-granting ticket
(3) C → TGS: Options ‖ ID_v ‖ $Times$ ‖ ‖ $Nonce_2$ ‖ $Ticket_{tgs}$ ‖ $Authenticator_c$
(4) TGS → C: $Realm_c \| ID_C \| Ticket_v \| E_{K_{c,tgs}}\left[K_{c,v} \| Times \| Nonce_2 \| Realm_v \| ID_V\right]$
$Ticket_{tgs} = E_{K_{tgs}}\left[Flags \| K_{c,tgs} \| Realm_c \| ID_C \| AD_C \| Times\right]$
$Ticket_v = E_{K_v}\left[Flags \| K_{c,v} \| Realm_c \| ID_C \| AD_C \| Times\right]$
$Authenticator_c = E_{K_{c,tgs}}\left[ID_C \| Realm_c \| TS_1\right]$
(c) Client/Server Authentication Exchange: to obtain service
(5) C → V: Options ‖ Ticket_v ‖ Authenticator_c
(6) V → C: $E_{K_{c,V}}$ [TS_2 ‖ Subkey ‖ Seq#]
$Ticket_v = E_{K_v}\left[Flags \| K_{c,v} \| Realm_c \| ID_C \| AD_C \| Times\right]$
$Authenticator_c = E_{K_{c,v}}\left[ID_C \| Realm_c \| TS_2 \| Subkey \| Seq\#\right]$

irst, consider the authentication service exchange. Message (1) is a client request for a ticket-granting ticket. As before, it includes the ID of the user and the TGS. The following new

elements are added:

Realm: Indicates realm of user

Options: Used to request that certain flags be set in the returned ticket

Times: Used by the client to request the following time settings in the ticket:

from: the desired start time for the requested ticket

till: the requested expiration time for the requested ticket rtime: requested renew-till time

Nonce: A random value to be repeated in message (2) to assure that the response is fresh and has not been replayed by an opponent

Message (2) returns a ticket-granting ticket, identifying information for the client, and a block encrypted using the encryption key based on the user's password.
This block includes the session key to be used between the client and the TGS, times specified in message (1), the nonce from message (1), and TGS identifying information.

The ticket itself includes the session key, identifying information for the client, the requested time values, and flags that reflect the status of this ticket and the requested options.

These flags introduce significant new functionality to version 5. For now, we defer a discussion of these flags and concentrate on the overall structure of the version 5 protocol.

Let us now compare the ticket-granting service exchange for versions 4 and 5. We see that message (3) for both versions includes an authenticator, a ticket, and the name of the requested service.

In addition, version 5 includes requested times and options for the ticket and a nonce, all with functions similar to those of message (1).

The authenticator itself is essentially the same as the one used in version 4.

Message (4) has the same structure as message (2), returning a ticket plus information needed by

the client, the latter encrypted with the session key now shared by the client and the TGS.

Finally, for the client/server authentication exchange, several new features appear in version 5. In message (5), the client may request as an option that mutual authentication is required. The authenticator includes several new fields as follows:

Subkey: The client's choice for an encryption key to be used to protect this specific application session. If this field is omitted, the session key from the ticket ($K_{c,v}$) is used.

Sequence number: An optional field that specifies the starting sequence number to be use may be sequence numbered to detect replays.

If mutual authentication is required, the server responds with message (6). This message includes the timestamp from the authenticator. Note that in version 4, the timestamp was incremented by one. This is not necessary in version 5 because the nature of the format of messages is such that it is not possible for an opponent to create message (6) without knowledge of the appropriate encryption keys.

Ticket Flags

The flags field included in tickets in version 5 supports expanded functionality compared to that available in version 4.

X.509 Certificates

Overview:

- **issued by a Certification Authority (CA), containing:**

– version (1, 2, or 3)

– serial number (unique within CA) identifying certificate

– signature algorithm identifier

– issuer X.500 name (CA)

– period of validity (from - to dates)

– subject X.500 name (name of owner)

– subject public-key info (algorithm, parameters, key)

– issuer unique identifier (v2+)

– subject unique identifier (v2+)

– extension fields (v3)

– signature (of hash of all fields in certificate)

- **notation CA<<A>> denotes certificate for A signed by CA**

X.509 defines a framework for the provision of authentication services by the X.500 directory to its users. The directory may serve as a repository of public-key certificates. Each certificate contains the public key of a user and is signed with the private key of a trusted certification authority. In addition, X.509 defines alternative authentication protocols based on the use of public-key certificates.

X.509 is an important standard because the certificate structure and authentication protocols defined in X.509 are used in a variety of contexts. For example, the X.509 certificate format is used in S/MIME), IP Security and SSL/TLS and SET

X.509 is based on the use of public-key cryptography and digital signatures. The standard does not

dictate the use of a specific algorithm but recommends RSA. The digital signature scheme is assumed to require the use of a hash function.

Certificates

The heart of the X.509 scheme is the public-key certificate associated with each user. These user certificates are assumed to be created by some trusted certification authority (CA) and placed in the directory by the CA or by the user.

Version:

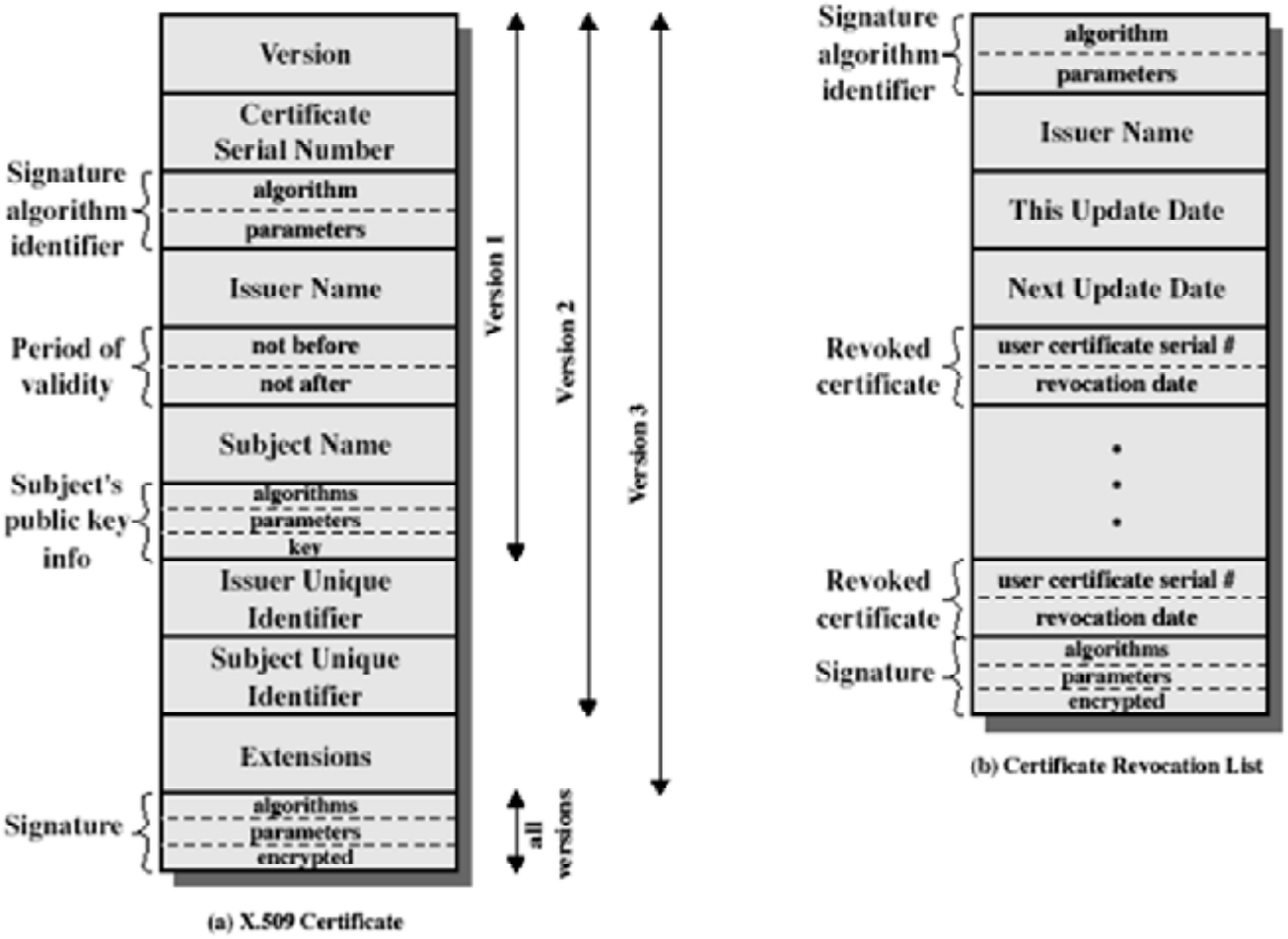
Differentiates among successive versions of the certificate format; the default is version 1. If the Issuer Unique Identifier or Subject Unique Identifier are present, the value must be version 2. If one or more extensions are present, the version must be version 3.

Serial number:

An integer value, unique within the issuing CA, that is unambiguously associated with this certificate.

Signature algorithm identifier:

The algorithm used to sign the certificate, together with any associated parameters. Because this information is repeated in the Signature field at the end of the certificate, this field has little, if any, utility.



Issuer name:

X.500 name of the CA that created and signed this certificate.

Period of validity:

Consists of two dates: the first and last on which the certificate is valid.

Subject name:

The name of the user to whom this certificate refers. That is, this certificate certifies the public key of the subject who holds the corresponding private key.

Subject's public-key information:

The public key of the subject, plus an identifier of the algorithm for which this key is to be used, together with any associated parameters.

Issuer unique identifier:

An optional bit string field used to identify uniquely the issuing CA in the event the X.500 name has been reused for different entities.

Subject unique identifier:

An optional bit string field used to identify uniquely the subject in the event the X.500 name has been reused for different entities.

Extensions:

A set of one or more extension fields. Extensions were added in version 3 and are discussed later in this section.

Signature:

Covers all of the other fields of the certificate; it contains the hash code of the other fields, encrypted with the CA's private key. This field includes the signature algorithm identifie

The standard uses the following notation to define a certificate: CA<<A>> = CA {V, SN, AI, CA, T_A, A, Ap} where

Y <<X>> = the certificate of user X issued by certification authority Y Y {I} = the signing of I by Y. It consists of I with an encrypted hash code appended

The CA signs the certificate with its private key. If the corresponding public key is known to a user, then that user can verify that a certificate signed by the CA is valid.

Obtaining a User's Certificate

User certificates generated by a CA have the following characteristics:

Any user with access to the public key of the CA can verify the user public key that was certified.

No party other than the certification authority can modify the certificate without this being detected.

ecause certificates are unforgeable, they can be placed in a directory without the need for the directory to make special efforts to protect them.

If all users subscribe to the same CA, then there is a common trust of that CA. All user certificates can be placed in the directory for access by all users.

If there is a large community of users, it may not be practical for all users to subscribe to the same CA. Because it is the CA that signs certificates, each participating user must have a copy of the CA's own public key to verify signatures. This public key must be provided to each user in an absolutely secure (with respect to integrity and authenticity) way so that the user has confidence in the associated certificates. Thus, with many users, it may be more practical for there to be a number of CAs, each of which securely provides its public key to some fraction of the users.

Now suppose that A has obtained a certificate from certification authority X_1 and B has obtained a certificate from CA X_2. If A does not securely know the public key of X_2, then B's certificate, issued by X_2, is useless to A.

A can read B's certificate, but A cannot verify the signature. However, if the two CAs have securely exchanged their own public keys, the following procedure will enable A to obtain B's public key:

1. A obtains, from the directory, the certificate of X_2 signed by X_1. Because A securely knows X_1's public key, A can obtain X_2's public key from its certificate and verify it by means of X_1's signature on the certificate.
2. A then goes back to the directory and obtains the certificate of B signed by X_2

Because A now has a trusted copy of X_2's public key, A can verify the signature and securely obtain B's public key.

A has used a chain of certificates to obtain B's public key. In the notation of X.509, this chain is expressed as

$X_1\langle\langle X_2\rangle\rangle$ $X_2\langle\langle B\rangle\rangle$

In the same fashion, B can obtain A's public key with the reverse chain: $X_2\langle\langle X_1\rangle\rangle$ $X_1\langle\langle A\rangle\rangle$

This scheme need not be limited to a chain of two certificates. An arbitrarily long path of CAs can be followed to produce a chain. A chain with N elements would be expressed as

$X_1\langle\langle X_2\rangle\rangle$ $X_2\langle\langle X_3\rangle\rangle$... $X_N\langle\langle B\rangle\rangle$

In this case, each pair of CAs in the chain (X_i, X_{i+1}) must have created certificates for each other.

All these certificates of CAs by CAs need to appear in the directory, and the user needs to know how they are linked to follow a path to another user's public-key certificate. X.509 suggests that CAs be arranged in a hierarchy so that navigation is straightforward.

Figure 14.5, taken from X.509, is an example of such a hierarchy. The connected circles indicate the hierarchical relationship among the CAs; the associated boxes indicate certificates maintained in the directory for each CA entry. The directory entry for each CA includes two types of certificates:

Forward certificates: Certificates of X generated by other CAs

Reverse certificates: Certificates generated by X that are the certificates of other CAs

CA Hierarchy Use

In the example given below , user A can acquire the following certificates from the directory to establish a certification path to B:

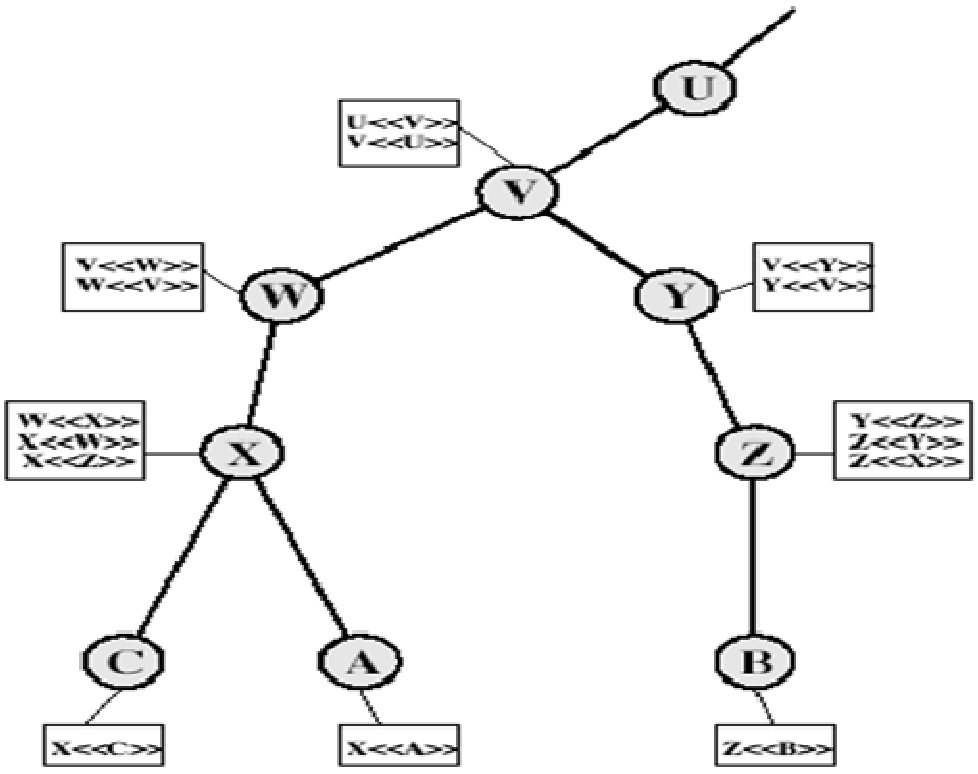
X<<W>> W <<V>> V <<Y>> <<Z>> Z <<B>>

When A has obtained these certificates, it can unwrap the certification path in sequence to recover a trusted copy of B's public key. Using this public key, A can send encrypted

Messages to B. If A wishes to receive encrypted messages back from B, or to sign messages sent to B, then B will require A's public key, which can be obtained from the following certification path:

Z<<Y>> Y <<V>> V <<W>> W <<X>>X <<A>>

B can obtain this set of certificates from the directory, or A can provide them as part of its initial message to B.



Certificate Revocation

- Certificates have a period of validity
- may need to revoke before expiry, for the following reasons eg:

1. user's private key is compromised
2. User is no longer certified by this CA

3. CA's certificate is compromised

- CA's maintain list of revoked certificates

1. the Certificate Revocation List (CRL)

- users should check certs with CA's CRL

Authentication Procedures

X.509 includes three alternative authentication procedures:

- **One-Way Authentication**
- **Two-Way Authentication**
- **Three-Way Authentication**
- all use public-key signatures

One-Way Authentication

- 1 message (A->B) used to establish
  - the identity of A and that message is from A
  - message was intended for B
  - integrity & originality of message
- message must include timestamp, nonce, B's identity and is signed by A

Two-Way Authentication

- 2 messages (A->B, B->A) which also establishes in addition:
  - the identity of B and that reply is from B
  - that reply is intended for A
  - integrity & originality of reply

  reply includes original nonce from A, also timestamp and nonce from B

Three-Way Authentication

- 3 messages (A->B, B->A, A->B) which enables above authentication without synchronized clocks
- has reply from a back to B containing signed copy of nonce from B
- means that timestamps need not be checked or relied upon

X.509 Version 3

The X.509 version 2 format does not convey all of the information that recent design and implementation experience has shown to be needed. [FORD95] lists the following requirements not satisfied by version 2:

1. The Subject field is inadequate to convey the identity of a key owner to a public- key user.
2. The Subject field is also inadequate for many applications, which typically recognize entities by an Internet e-mail address, a URL, or some other Internet- related identification.
3. There is a need to indicate security policy information. There is a need to limit the damage that can result from a faulty or malicious CA by setting constraints on the applicability of a particular certificate.
4. It is important to be able to identify different keys used by the same owner at different times.

The certificate extensions fall into three main categories: key and policy information, subject and issuer attributes, and certification path constraints.

Key and Policy Information

These extensions convey additional information about the subject and issuer keys, plus indicators of certificate policy.. For example, a policy might be applicable to the authentication of electronic data interchange (EDI) transactions for the trading of goods within a given price range.

This area includes the following:

Authority key identifier: Identifies the public key to be used to verify the signature on this certificate or CRL.

Subject key identifier: Identifies the public key being certified. Useful for subject key pair updating.

Key usage: Indicates a restriction imposed as to the purposes for which, and the policies under which, the certified public key may be used.

Private-key usage period: Indicates the period of use of the private key corresponding to the public key.. For example, with digital signature keys, the usage period for the signing private key is typically shorter than that for the verifying public key.

Certificate policies: Certificates may be used in environments where multiple policies apply.

Policy mappings: Used only in certificates for CAs issued by other CAs.

Certificate Subject and Issuer Attributes

These extensions support alternative names, in alternative formats, for a certificate subject or certificate issuer and can convey additional information about the certificate subject, to increase a certificate user's confidence that the certificate subject is a particular person or entity. For example, information such as postal address, position within a corporation, or picture image may be required. The extension fields in this area include the following:

Subject alternative name: Contains one or more alternative names, using any of a variety of forms

Subject directory attributes: Conveys any desired X.500 directory attribute values for the subject of this certificate.

Certification Path Constraints

These extensions allow constraint specifications to be included in certificates issued for CAs by other CAs.The extension fields in this area include the following:

Basic constraints: Indicates if the subject may act as a CA. If so, a certification path length constraint may be specified.

Name constraints: Indicates a name space within which all subject names in subsequent certificates in a certification path must be located.

Policy constraints: Specifies constraints that may require explicit certificate policy identification or inhibit policy mapping for the remainder of the certification path.

ELECTRONIC MAIL SECURITY PRETTY GOOD PRIVACY (PGP)

PGP provides the confidentiality and authentication service that can be used for electronic mail and file storage applications. The steps involved in PGP are

Select the best available cryptographic algorithms as building blocks.

Integrate these algorithms into a general purpose application that is independent of operating system and processor and that is based on a small set of easy-to-use commands.

Make the package and its documentation, including the source code, freely available via the internet, bulletin boards and commercial networks.

Enter into an agreement with a company to provide a fully compatible, low cost

commercial version of PGP.

PGP has grown explosively and is now widely used. A number of reasons can be cited for this growth.

- It is available free worldwide in versions that run on a variety of platform.
- It is based on algorithms that have survived extensive public review and are considered extremely secure.
- e.g., RSA, DSS and Diffie Hellman for public key encryption CAST-128, IDEA and 3DES for conventional encryption SHA-1 for hash coding.
- it has a wide range of applicability.
- It was not developed by, nor it is controlled by, any governmental or standards organization.

Operational description

The actual operation of PGP consists of five services: authentication, confidentiality, compression, e-mail compatibility and segmentation.

1. Authentication

The sequence for authentication is as follows:

The sender creates the message

SHA-1 is used to generate a 160-bit hash code of the message

The hash code is encrypted with RSA using the sender's private key and the result is prepended to the message

The receiver uses RSA with the sender's public key to decrypt and recover the hash code.

The receiver generates a new hash code for the message and compares it with the decrypted hash code. If the two match, the message is accepted as authentic.

2. Confidentiality

Confidentiality is provided by encrypting messages to be transmitted or to be stored locally as files. In both cases, the conventional encryption algorithm CAST-128 may be used. The 64-bit cipher feedback (CFB) mode is used.

In PGP, each conventional key is used only once. That is, a new key is generated as a random 128-bit number for each message. Thus although this is referred to as **a session key**, it is in reality a **one time key**. To protect the key, it is encrypted with the receiver's public key.

The sequence for confidentiality is as follows:

- The sender generates a message and a random 128-bit number to be used as a session key for this message only.
- The message is encrypted using CAST-128 with the session key.
- The session key is encrypted with RSA, using the receiver's public key and is prepended to the message.
- The receiver uses RSA with its private key to decrypt and recover the session key.
- The session key is used to decrypt the message.

Confidentiality and authentication

Here both services may be used for the same message. First, a signature is generated for the plaintext message and prepended to the message. Then the plaintext plus the signature is encrypted using CAST-128 and the session key is encrypted using RSA.

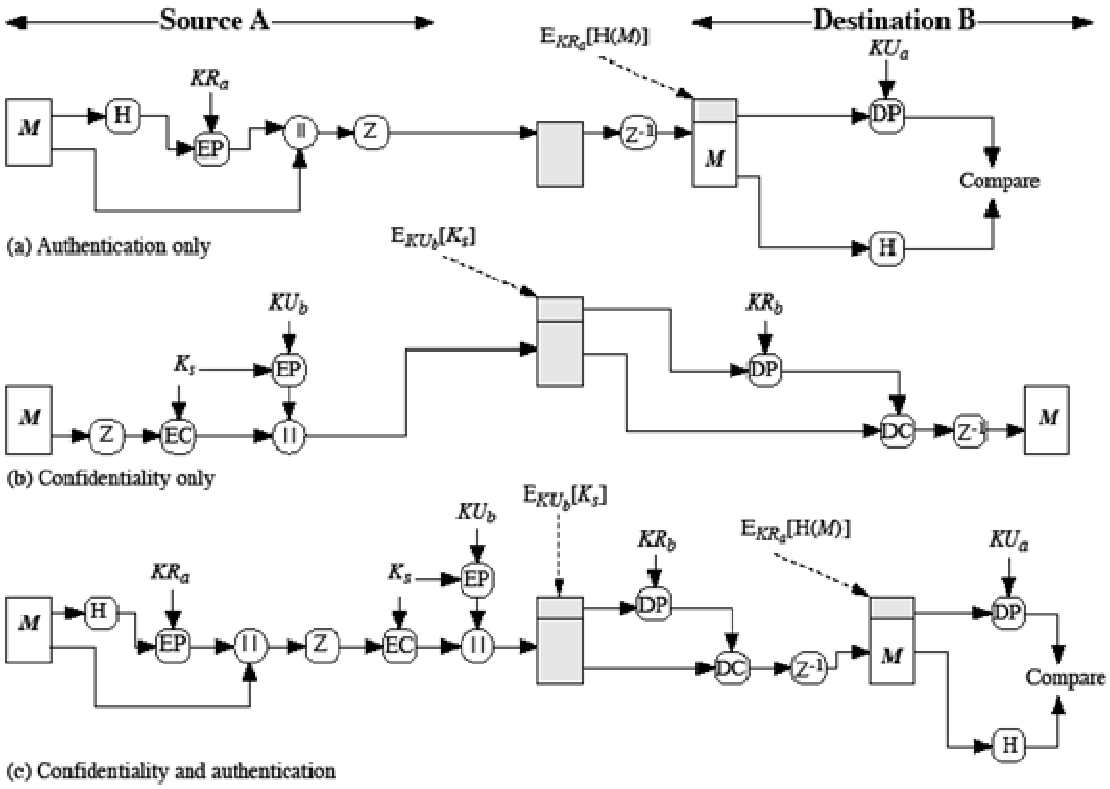


Figure 15.1 PGP Cryptographic Functions

3. Compression

As a default, PGP compresses the message after applying the signature but before encryption. This has the benefit of saving space for both e-mail transmission and for file storage.

The signature is generated before compression for two reasons:

It is preferable to sign an uncompressed message so that one can store only the uncompressed message together with the signature for future verification. If one signed a compressed document, then it would be necessary either to store a compressed version of the message for later verification or to recompress the message when verification is required.

Even if one were willing to generate dynamically a recompressed message fro verification, PGP's compression algorithm presents a difficulty. The algorithm is not deterministic; various implementations of the algorithm achieve different tradeoffs in running speed versus compression ratio and as a result, produce different compression forms.

Message encryption is applied after compression to strengthen cryptographic security. Because the compressed message has less redundancy than the original plaintext, cryptanalysis is more difficult.

The compression algorithm used is ZIP.

4. e-mail compatibility

Many electronic mail systems only permit the use of blocks consisting of ASCII texts. To accommodate this restriction, PGP provides the service of converting the raw 8-bit binary stream to a stream of printable ASCII characters. The scheme used for this purpose is **radix-64 conversion**. Each group of three octets of binary data is mapped into four ASCII characters.

e.g., consider the 24-bit (3 octets) raw text sequence 00100011 01011100 10010001, we can express this input in block of 6-bits to produce 4 ASCII characters.

001000 110101 110010 010001

I L Y R => corresponding ASCII characters

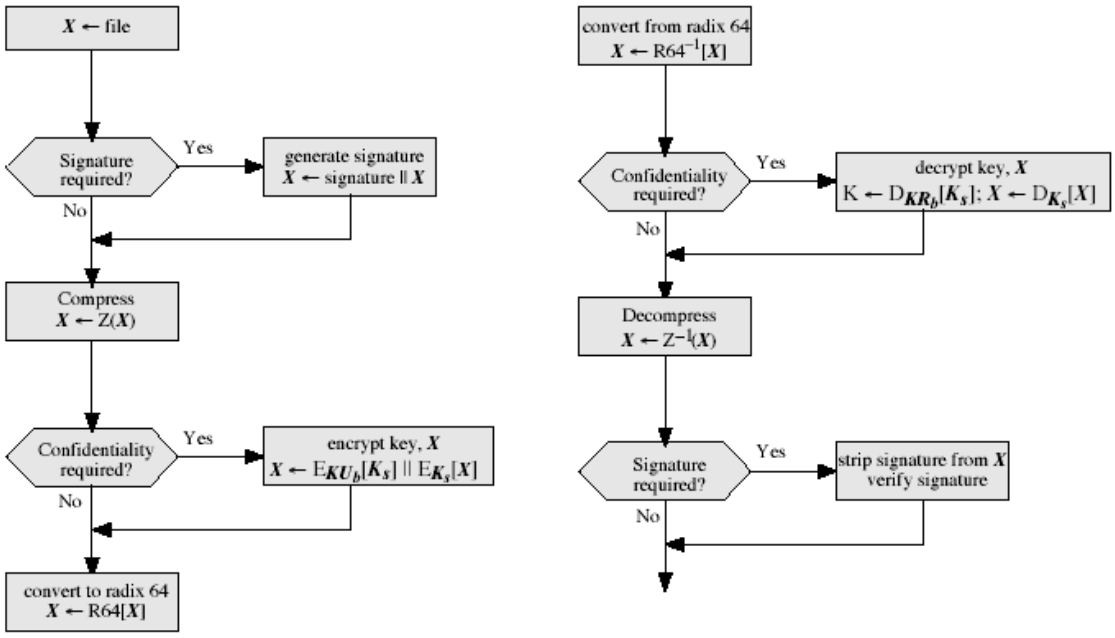
5. Segmentation and reassembly

E-mail facilities often are restricted to a maximum length. E.g., many of the facilities accessible through the internet impose a maximum length of 50,000 octets. Any message longer than that must be broken up into smaller segments, each of which is mailed separately.

To accommodate this restriction, PGP automatically subdivides a message that is too large into

segments that are small enough to send via e-mail. The segmentation is done after all the other processing, including the radix-64 conversion. At the receiving end, PGP must strip off all e-mail headers and reassemble the entire original block before performing the other steps.

PGP Operation Summary:



(a) Generic Transmission Diagram (from A)

(b) Generic Reception Diagram (to B)

Cryptographic keys and key rings

Three separate requirements can be identified with respect to these keys:

A means of generating unpredictable session keys is needed.

It must allow a user to have multiple public key/private key pairs.

Each PGP entity must maintain a file of its own public/private key pairs as well as a file of public keys of correspondents.

We now examine each of the requirements in turn.

1. Session key generation

Each session key is associated with a single message and is used only for the purpose of encryption and decryption of that message. Random 128-bit numbers are generated using CAST-128 itself. The input to the random number

generator consists of a 128-bit key and two 64-bit blocks that are treated as plaintext to be encrypted. Using cipher feedback mode, the CAST-128 produces two 64-bit cipher text blocks, which are concatenated to form the 128-bit session key. The plaintext input to CAST-128 is itself derived from a stream of 128-bit randomized numbers. These numbers are based on the keystroke input from the user.

2. Key identifiers

If multiple public/private key pair are used, then how does the recipient know which of the public keys was used to encrypt the session key? One simple solution would be to transmit the public key with the message but, it is unnecessary wasteful of space. Another solution would be to associate an identifier with each public key that is unique at least within each user.

The solution adopted by PGP is to assign a key ID to each public key that is, with very high probability, unique within a user ID. The key ID associated with each public key consists of its least significant 64 bits. i.e., the key ID of public key KU_a is ($KU_a \bmod 2^{64}$).

A message consists of three components.

Message component – includes actual data to be transmitted, as well as the filename and a timestamp that specifies the time of creation.

Signature component – includes the following

- Timestamp – time at which the signature was made.
- Message digest – hash code.

O Two octets of message digest – to enable the recipient to determine if the correct public key was used to decrypt the message.

- Key ID of sender's public key – identifies the public key

Session key component – includes session key and the identifier of the recipient public key.

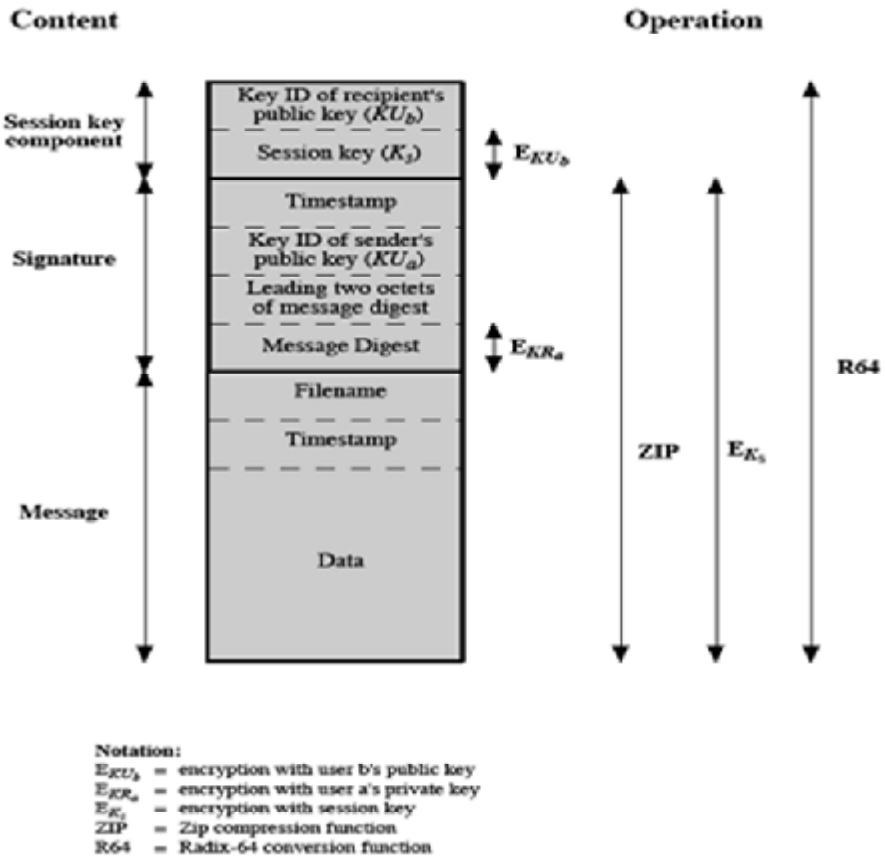


Figure 15.3 General Format of PGP Message (from A to B)

3. Key rings

PGP provides a pair of data structures at each node, one to store the public/private key pair owned by that node and one to store the public keys of the other users known at that node. These data structures are referred to as private key ring and public key ring.

The general structures of the private and public key rings are shown below: Timestamp – the date/time when this entry was made.

Key ID – the least significant bits of the public key.

Public key – public key portion of the pair. **Private key** – private key portion of the pair. **User ID** – the owner of the key.

Key legitimacy field – indicates the extent to which PGP will trust that this is a valid public key for this user.

Private Key Ring

Timestamp	Key ID*	Public Key	Encrypted Private Key	User ID*
• • •	• • •	• • •	• • •	• • •
T_i	PU_i mod 2^{64}	PU_i	E(H(P_i), PR_i)	User i
• • •	• • •	• • •	• • •	• • •

Public Key Ring

Timestamp	Key ID*	Public Key	Owner Trust	User ID*	Key Legitimacy	Signature(s)	Signature Trust(s)
• • •	• • •	• • •	• • •	• • •	• • •	• • •	• • •
T_i	PU_i mod 2^{64}	PU_i	trust_flag$_i$	User i	trust_flag$_i$		
• • •	• • •	• • •	• • •	• • •	• • •	• • •	• • •

* = field used to index table

Figure 15.4 General Structure of Private and Public Key Rings

Signature trust field – indicates the degree to which this PGP user trusts the signer to certify public key.

Owner trust field – indicates the degree to which this public key is trusted to sign other public key certificates.

PGP message generation

First consider message transmission and assume that the message is to be both signed and encrypted. The sending PGP entity performs the following steps:

1. **signing the message**

PGP retrieves the sender's private key from the private key ring using user ID as an index. If user ID was not provided, the first private key from the ring is retrieved.

PGP prompts the user for the passpharse (password) to recover the unencrypted private key.

The signature component of the message is constructed.

2. **encrypting the message**

PGP generates a session key and encrypts the message.

PGP retrieves the recipient's public key from the public key ring using user

ID as index.

The session key component of the message is constructed. The receiving PGP entity performs the following steps:

Decrypting the message

PGP retrieves the receiver's private key from the private key ring, using the key ID field in the session key component of the message as an index.

PGP prompts the user for the passpharse (password) to recover the unencrypted private key.

PGP then recovers the session key and decrypts the message.

2. **Authenticating the message**

PGP retrieves the sender's public key from the public key ring, using the key ID field in the signature key component of the message as an index.

PGP recovers the transmitted message digest.

PGP computes the message digest for the received message and compares it to the transmitted message digest to authenticate.

Public-Key Management

This whole business of protecting public keys from tampering is the single most difficult problem in practical public key applications. PGP provides a structure for solving this problem, with several suggested options that may be used.

Approaches to Public-Key Management

The essence of the problem is this: User A must build up a public-key ring containing the public keys of other users to interoperate with them using PGP. Suppose that A's key ring contains a public key attributed to B but that the key is, in fact, owned by C. This could happen if, for

example, A got the key from a bulletin board system (BBS) that was used by B to post the public key but that has been compromised by C. The result is that two threats now exist. First, C can send messages to A and forge B's signature, so that A will accept the message as coming from B. Second, any encrypted message from A to B can be read by C.

A number of approaches are possible for minimizing the risk that a user's public-key ring contains false public keys. Suppose that A wishes to obtain a reliable public key for B. The following are some approaches that could be used:

1. Physically get the key from B. B could store her public key (PU_b) on a floppy disk and hand it to A.
2. Verify a key by telephone. If A can recognize B on the phone, A could call B and ask her to dictate the key, in radix-64 format, over the phone.
3. Obtain B's public key from a mutual trusted individual D. For this purpose, the introducer, D, creates a signed certificate. The certificate includes B's public key, the time of creation of the key, and a validity period for the key.
4. Obtain B's public key from a trusted certifying authority. Again, a public key certificate is created and signed by the authority. A could then access the authority, providing a user name and receiving a signed certificate.

For cases 3 and 4, A would already have to have a copy of the introducer's public key and trust that this key is valid. Ultimately, it is up to A to assign a level of trust to anyone who is to act as an introducer.

The Use of Trust

Although PGP does not include any specification for establishing certifying authorities or for establishing trust, it does provide a convenient means of using trust, associating trust with public keys, and exploiting trust information.

The basic structure is as follows. Each entry in the public-key ring is a public-key certificate. Associated with each such entry is a key legitimacy field that indicates the extent to which PGP will trust that this is a valid public key for this user; the higher the level of trust, the stronger is the binding of this user ID to this key.

This field is computed by PGP.Also associated with the entry are zero or more signatures that the key ring owner has collected that sign this certificate. In turn, each signature has associated

with it a signature trust field that indicates the degree to which this PGP user trusts the signer to certify public keys.The key legitimacy field is derived from the collection of signature trust fields in the entry.Finally, each entry defines a public key associated with a particular owner, and an owner trust field is included that indicates the degree to which this public key is trusted to sign other public-key certificates; this level of trust is assigned by the user.

The three fields mentioned in the previous paragraph are each contained in a structure referred to as a trust flag byte.

Suppose that we are dealing with the public-key ring of user A. We can describe the operation of the trust processing as follows:

1. When A inserts a new public key on the public-key ring, PGP must assign a value to the trust flag that is associated with the owner of this public key. If the owner is A, and therefore this public key also appears in the private-key ring, then a value of ultimate trust is automatically assigned to the trust field. Otherwise, PGP asks A for his assessment of the trust to be assigned to the owner of this key, and A must enter the desired level. The user can specify that this owner is unknown, untrusted, marginally trusted, or completely trusted.
2. When the new public key is entered, one or more signatures may be attached to it.

More signatures may be added later. When a signature is inserted into the entry, PGP searches the public-key ring to see if the author of this signature is among the known public-key owners. If so, the OWNERTRUST value for this owner is assigned to the SIGTRUST field for this signature. If not, an unknown user value is assigned.

3. The value of the key legitimacy field is calculated on the basis of the signature trust fields present in this entry. If at least one signature has a signature trust value of ultimate, then the key legitimacy value is set to complete.

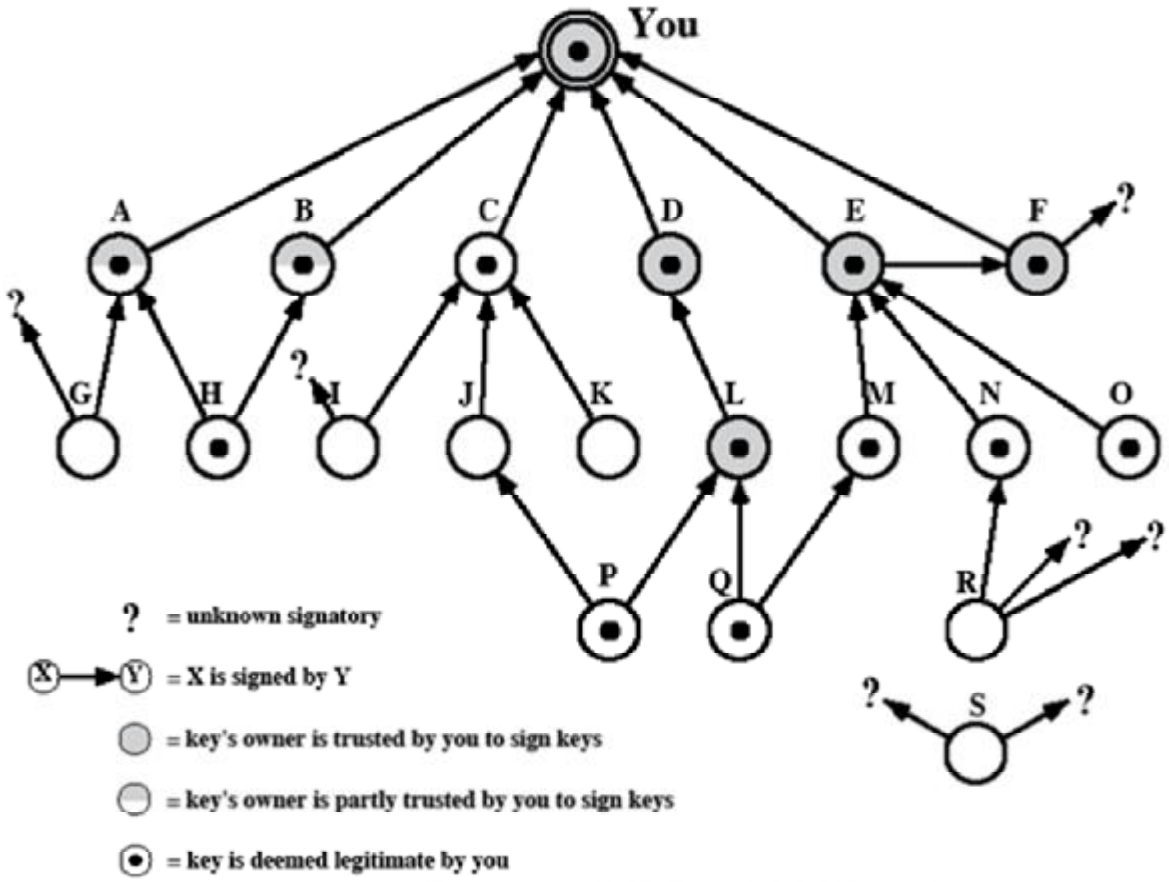


Figure 15.7 PGP Trust Model Example

The node labeled "You" refers to the entry in the public-key ring corresponding to this user. This key is legitimate and the OWNERTRUST value is ultimate trust. Each other node in the key ring has an OWNERTRUST value of undefined unless some other value is assigned by the user. In this example, this user has specified that it always trusts the following users to sign other keys: D, E, F, L. This user partially trusts users A and B to sign other keys.

So the shading, or lack thereof, of the nodes in Figure 15.7 indicates the level of trust assigned by this user. The tree structure indicates which keys have been signed by which

other users. If a key is signed by a user whose key is also in this key ring, the arrow joins the signed key to the signatory. If the key is signed by a user whose key is not present in this key ring, the arrow joins the signed key to a question mark, indicating that the signatory is unknown to this user.

Note that all keys whose owners are fully or partially trusted by this user have been signed by this user, with the exception of node L.

1. We assume that two partially trusted signatures are sufficient to certify a key. Hence, the key for user H is deemed legitimate by PGP because it is signed by A and B, both of whom are partially trusted.
2. A key may be determined to be legitimate because it is signed by one fully trusted or two partially trusted signatories, but its user may not be trusted to sign other keys. For example, N's key is legitimate because it is signed by E, whom this user trusts, but N is not trusted to sign other keys because this user has not assigned N that trust value. Therefore, although R's key is signed by N, PGP does not consider R's key legitimate. This situation makes perfect sense. If you wish to send a private message to some individual, it is not necessary that you trust that individual in any respect. It is only necessary that you are sure that you have the correct public key for that individual.
3. Figure 15.7 also shows an example of a detached "orphan" node S, with two unknown signatures. Such a key may have been acquired from a key server. PGP cannot assume that this key is legitimate simply because it came from a reputable server. The user must declare the key legitimate by signing it or by telling PGP that it is willing to trust fully one of the key's signatories.

S/MIME

S/MIME (Secure/Multipurpose Internet Mail Extension) is a security enhancement to the MIME Internet e-mail format standard, based on technology from RSA Data Security. S/MIME is defined in a number of documents, most importantly RFCs 3369, 3370, 3850 and 3851.

Multipurpose Internet Mail Extensions

MIME is an extension to the RFC 822 framework that is intended to address some of the problems and limitations of the use of SMTP (Simple Mail Transfer Protocol) or some other mail transfer protocol and RFC 822 for electronic mail. Following are the limitations of SMTP/822 scheme:

1. SMTP cannot transmit executable files or other binary objects.
2. SMTP cannot transmit text data that includes national language characters because these are represented by 8-bit codes with values of 128 decimal or higher, and SMTP is limited to 7-bit ASCII.
3. SMTP servers may reject mail message over a certain size.
4. SMTP gateways that translate between ASCII and the character code EBCDIC do not use a consistent set of mappings, resulting in translation problems.
5. SMTP gateways to X.400 electronic mail networks cannot handle nontextual data included in X.400 messages.
6. Some SMTP implementations do not adhere completely to the SMTP standards defined in RFC 821. Common problems include:

- o Deletion, addition, or reordering of carriage return and linefeed
- o Truncating or wrapping lines longer than 76 characters
- o Removal of trailing white space (tab and space characters)
- o Padding of lines in a message to the same length
- o Conversion of tab characters into multiple space characters

MIME is intended to resolve these problems in a manner that is compatible with existing RFC 822 implementations. The specification is provided in RFCs 2045 through 2049.

Overview

The MIME specification includes the following elements:

1. **Five new message header** fields are defined, which may be included in an RFC 822 header. These fields provide information about the body of the message.
2. **A number of content formats** are defined, thus standardizing representations that support multimedia electronic mail.
3. **Transfer encodings** are defined that enable the conversion of any content format into a form that is protected from alteration by the mail system.

In this subsection, we introduce the five message header fields. The next two subsections deal with content formats and transfer encodings.

The five header fields defined in MIME are as follows:

MIME-Version: Must have the parameter value 1.0. This field indicates that the message conforms to RFCs 2045 and 2046.

Content-Type: Describes the data contained in the body with sufficient detail

Content-Transfer-Encoding: Indicates the type of transformation that has been used to represent the body of the message in a way that is acceptable for mail transport.

Content-ID: Used to identify MIME entities uniquely in multiple contexts.

Content-Description: A text description of the object with the body; this is useful when the object is not readable (e.g., audio data).

MIME Content Types

The bulk of the MIME specification is concerned with the definition of a variety of content types. This reflects the need to provide standardized ways of dealing with a wide variety of information representations in a multimedia environment.

Table 15.3 lists the content types specified in RFC 2046. There are seven different major types of content and a total of 15 subtypes

Type	Subtype	Description
Text	Plain	Unformatted text; may be ASCII or ISO 8859.
	Enriched	Provides greater format flexibility.
Multipart	Mixed	The different parts are independent but are to be transmitted together. They should be presented to the receiver in the order that they appear in the mail message.
	Parallel	Differs from Mixed only in that no order is defined for delivering the parts to the receiver.
	Alternative	The different parts are alternative versions of the same information. They are ordered in increasing faithfulness to the original, and the recipient's mail system should display the
	Digest	Similar to Mixed, but the default type/subtype of each part is message/rfc822.
Message	rfc822	The body is itself an encapsulated message that conforms to RFC 822.
	Partial	Used to allow fragmentation of large mail items, in a way that is transparent to the recipient.
	External-body	Contains a pointer to an object that exists elsewhere.
Image	jpeg	The image is in JPEG format, JFIF encoding.

	gif	The image is in GIF format.
Video	mpeg	MPEG format.
Audio	Basic	Single-channel 8-bit ISDN mu-law encoding at a sample rate of 8 kHz.
Application	PostScript	Adobe Postscript.
	octet- stream	General binary data consisting of 8-bit bytes.

For the text type of body, no special software is required to get the full meaning of the text, aside from support of the indicated character set. The primary subtype is plain text, which is simply a string of ASCII characters or ISO 8859 characters. The enriched subtype allows greater formatting flexibility.

The multipart type indicates that the body contains multiple, independent parts. The Content-Type header field includes a parameter, called boundary, that defines the delimiter between body parts. The multipart/digest subtype is used when each of the body parts is interpreted as an RFC 822 message with headers. This subtype enables the construction of a message whose parts are individual messages. For example, the moderator of a group might collect e- mail messages from participants, bundle these messages, and send them out in one encapsulating MIME message.

The message type provides a number of important capabilities in MIME. The message/rfc822 subtype indicates that the body is an entire message, including header and body. Despite the name of this subtype, the encapsulated message may be not only a simple RFC 822 message, but also any MIME message.

The message/partial subtype enables fragmentation of a large message into a number of parts, which must be reassembled at the destination. For this subtype, three parameters are specified in the Content-Type: Message/Partial field: an id common to all fragments of the same message, a sequence number unique to each fragment, and the total number of fragments.

the message/external-body subtype indicates that the actual data to be conveyed in this message are not contained in the body. Instead, the body contains the information needed to access the data. As with the other message types, the message/external-body subtype has an outer header and an encapsulated message with its own header. The only necessary field in the outer header is the Content-Type field, which identifies this as a message/external-body subtype. The inner header is the message header for

the encapsulated message. The Content-Type field in the outer header must include an access-type parameter, which indicates the method of access, such as FTP (file transfer protocol).

The application type refers to other kinds of data, typically either uninterpreted binary data or information to be processed by a mail-based application.

MIME Transfer Encodings

The other major component of the MIME specification, in addition to content type specification, is a definition of transfer encodings for message bodies. The objective is to provide reliable delivery across the largest range of environments.

The MIME standard defines two methods of encoding data. The Content-Transfer- Encoding field can actually take on six values, as listed in Table 15.4. For SMTP transfer, it is safe to use the 7bit form. The 8bit and binary forms may be usable in other mail transport contexts. Another Content-Transfer-Encoding value is x-token, which indicates that some other encoding scheme is used, for which a name is to be supplied. The two actual encoding schemes defined are quoted-printable and base64.

MIME Transfer Encodings

7bit	The data are all represented by short lines of ASCII characters.
8bit	The lines are short, but there may be non-ASCII characters (octets with the high-order bit set).
binary	Not only may non-ASCII characters be present but the lines are not necessarily short enough for SMTP transport.
quoted-printable	Encodes the data in such a way that if the data being encoded are mostly ASCII text, the encoded form of the data remains largely recognizable by .
base64	Encodes data by mapping 6-bit blocks of input to 8-bit blocks of output, all of which are printable ASCII characters.
x-token	A named nonstandard encoding.

The quoted-printable transfer encoding is useful when the data consists largely of octets that correspond to printable ASCII characters. In essence, it represents nonsafe characters by the hexadecimal representation of their code and introduces reversible (soft) line breaks to limit message lines to 76 characters.

The base64 transfer encoding, also known as radix-64 encoding, is a common one for encoding arbitrary binary data in such a way as to be invulnerable to the processing by mail transport programs.

Canonical Form

An important concept in MIME and S/MIME is that of canonical form. Canonical form is a format, appropriate to the content type that is standardized for use between systems. This is in contrast to native form, which is a format that may be peculiar to a particular system.

S/MIME Functionality

In terms of general functionality, S/MIME is very similar to PGP. Both offer the ability to sign and/or encrypt messages. In this subsection, we briefly summarize S/MIME capability. We then look in more detail at this capability by examining message formats and message preparation.

Functions

S/MIME provides the following functions:

Enveloped data: This consists of encrypted content of any type and encrypted- content encryption keys for one or more recipients.

Signed data: A digital signature is formed by taking the message digest of the content to be signed and then encrypting that with the private key of the signer. The content plus signature are then encoded using base64 encoding. A signed data message can only be viewed by a recipient with S/MIME capability.

Clear-signed data: As with signed data, a digital signature of the content is formed. However, in this case, only the digital signature is encoded using base64. As a result, recipients without S/MIME capability can view the message content, although they cannot verify the signature.

Signed and enveloped data: Signed-only and encrypted-only entities may be nested, so that encrypted data may be signed and signed data or clear-signed data may be encrypted.

Cryptographic Algorithms

- hash functions: SHA-1 & MD5
- digital signatures: DSS & RSA
- session key encryption: ElGamal & RSA
- message encryption: Triple-DES, RC2/40 and others
- have a procedure to decide which algorithms to use.

Table 15.6 summarizes the cryptographic algorithms used in S/MIME. S/MIME uses the following terminology, taken from RFC 2119 to specify the requirement level:

Must: The definition is an absolute requirement of the specification. An implementation must include this feature or function to be in conformance with the specification.

should: There may exist valid reasons in particular circumstances to ignore this feature or function, but it is recommended that an implementation include the feature or function.

S/MIME Messages

S/MIME makes use of a number of new MIME content types, which are shown in Table 15.7. All of the new application types use the designation PKCS. This refers to a set of public-key cryptography specifications issued by RSA Laboratories and made available for the S/MIME effort.

Function	Requirement
Create a message digest to be used in forming a digital signature. Encrypt message digest to form digital signature.	MUST support SHA-1. Receiver SHOULD support MD5 for backward compatibility. Sending and receiving agents MUST support DSS. Sending agents SHOULD support RSA encryption. Receiving agents SHOULD support verification of
Encrypt session key for transmission with message.	Sending and receiving agents SHOULD support Diffie-Hellman. Sending and receiving agents MUST support RSA
Encrypt message for transmission with one-time session key.	Sending and receiving agents MUST support encryption with triple DES Sending agents SHOULD support encryption with AES.

Create a message authentication code	Receiving agents MUST support HMAC with SHA-1. Rece ing a ents SHOULD support HMAC with

Type	Subtype	smime Parameter	Description
Multipart	Signed		A clear-signed message in two parts: one is the message and the other is the signature.
Application	pkcs 7- mime	signedData	A signed S/MIME entity.
	pkcs 7- mime	envelopedData	An encrypted S/MIME entity.
	pkcs 7- mime	degenerate si n dD ta	An entity containing only public- key certifi ates.
	pkcs 7- mime	CompressedData	A compressed S/MIME entity
	pkcs 7- si n ture	signedData	The content type of the signature subpart of a multi art/si ned messa

We examine each of these in turn after first looking at the general procedures for S/MIME message preparation.

SECURING A MIME ENTITY

S/MIME secures a MIME entity with a signature, encryption, or both. A MIME entity may be an entire message (except for the RFC 822 headers), or if the MIME content type is multipart, then a MIME entity is one or more of the subparts of the message. In all cases, the message to be sent is converted to canonical form. In particular, for a given type and subtype, the appropriate canonical form is used for the message content. For a multipart message, the appropriate canonical form is used

for each subpart.

The use of transfer encoding requires special attention.

i)EnvelopedData

An application/pkcs7-mime subtype is used for one of four categories of S/MIME processing, each with a unique smime-type parameter. In all cases, the resulting entity, referred to as an object, is represented in a form known as Basic Encoding Rules (BER), which is defined in ITU-T Recommendation X.209. The steps for preparing an envelopedData MIME entity are as follows:

1. Generate a pseudorandom session key for a particular symmetric encryption algorithm (RC2/40 or triple DES).

2. For each recipient, encrypt the session key with the recipient's public RSA key.

3. For each recipient, prepare a block known as Recipient Info that contains an identifier of the recipient's public-key certificate,[3] an identifier of the algorithm used to encrypt the session key, and the encrypted session key.

This is an X.509 certificate, discussed later in this section.

4. Encrypt the message content with the session key.

The RecipientInfo blocks followed by the encrypted content constitute the envelopedData. This information is then encoded into base64. To recover the encrypted message, the recipient first strips off the base64 encoding. Then the recipient's private key is used to recover the session key. Finally, the message content is decrypted with the session key.

ii)SignedData

The signedData smime-type can actually be used with one or more signers. For clarity, we confine our description to the case of a single digital signature. The steps for preparing a signedData MIME entity are as follows:

1. Select a message digest algorithm (SHA or MD5).

2. Compute the message digest, or hash function, of the content to be signed.

3. Encrypt the message digest with the signer's private key.

4. Prepare a block known as SignerInfo that contains the signer's public-key certificate, an identifier of the message digest algorithm, an identifier of the algorithm used to encrypt the message digest, and the encrypted message digest.

The signedData entity consists of a series of blocks, including a message digest algorithm identifier, the message being signed, and SignerInfo. The signedData entity may also include a set of public-key certificates sufficient to constitute a chain from a recognized root or top-level certification authority to the signer. This information is then encoded into base64.

To recover the signed message and verify the signature, the recipient first strips off the base64 encoding. Then the signer's public key is used to decrypt the message digest. The recipient independently computes the message digest and compares it to the decrypted message digest to verify the signature.

iii)Clear Signing

Clear signing is achieved using the multipart content type with a signed subtype.

As was mentioned, this signing process does not involve transforming the message to be signed, so that the message is sent "in the clear."

Thus, recipients with MIME capability but not S/MIME capability are able to read the incoming message.

A multipart/signed message has two parts. The first part can be any MIME type but must be prepared so that it will not be altered during transfer from source to destination. This means that if the first part is not 7bit, then it needs to be encoded using base64 or quoted- printable. Then this part is processed in the same manner as signedData, but in this case an object with signedData format is created that has an empty message content field. This object is a detached signature. It is then transfer encoded using base64 to become the second part of the multipart/signed message. This second part has a MIME content type of application and a subtype of pkcs7-signature

The protocol parameter indicates that this is a two-part clear-signed entity. The receiver can verify the signature by taking the message digest of the first part and comparing this to the message digest recovered from the signature in the second part.

Registration Request

Typically, an application or user will apply to a certification authority for a public-key certificate.

The application/pkcs10 S/MIME entity is used to transfer a certification request.

The certification request includes certificationRequestInfo block, followed by an identifier of the public-key encryption algorithm, followed by the signature of the certificationRequestInfo block, made using the sender's private key.

The certificationRequestInfo block includes a name of the certificate subject (the entity whose public key is to be certified) and a bit-string representation of the user's public key.

Certificates-Only Message

A message containing only certificates or a certificate revocation list (CRL) can be sent in response to a registration request. The message is an application/pkcs7-mime type/subtype with an smime-type parameter of degenerate. The steps involved are the same as those for creating a signedData message, except that there is no message content and the signerInfo field is empty.

S/MIME Certificate Processing

S/MIME uses public-key certificates that conform to version 3 of X.509 The key- management scheme used by S/MIME is in some ways a hybrid between a strict X.509 certification hierarchy and PGP's web of trust. As with the PGP model, S/MIME managers and/or users must configure each client with a list of trusted keys and with certificate revocation lists.

User Agent Role

An S/MIME user has **several key-management functions** to perform:

Key generation: The user of some related administrative utility (e.g., one associated with LAN management) MUST be capable of generating a key pair from a good source of nondeterministic random input and be protected in a secure fashion. A user agent SHOULD generate RSA key pairs with a length in the range of 768 to 1024 bits and MUST NOT generate a length of less than 512 bits.

Registration: A user's public key must be registered with a certification authority in order to receive an X.509 public-key certificate.

Certificate storage and retrieval: A user requires access to a local list of certificates in order to verify incoming signatures and to encrypt outgoing messages. Such a list could be maintained by the user or by some local administrative entity on behalf of a number of users.

VeriSign Certificates

There are several companies that provide certification authority (CA) services. For example, Nortel has designed an enterprise CA solution and can provide S/MIME support within an organization.

There are a number of Internet-based CAs, including VeriSign, GTE, and the U.S. Postal Service. Of these, the most widely used is the VeriSign CA service, a brief description of which we now provide.

VeriSign provides a CA service that is intended to be compatible with S/MIME and a variety of other applications. VeriSign issues X.509 certificates with the product name VeriSign Digital ID. As of early 1998, over 35,000 commercial Web sites were using VeriSign Server Digital IDs, and over a million consumer Digital IDs had been issued to users of Netscape and Microsoft browsers.

The information contained in a Digital ID depends on the type of Digital ID and its use. At a minimum, each Digital ID contains

Owner's public key

Owner's name or alias

Expiration date of the Digital ID

Serial number of the Digital ID

Name of the certification authority that issued the Digital ID

Digital signature of the certification authority that issued the Digital ID

Digital IDs can also contain other user-supplied information, including

Address

E-mail address

Basic registration information (country, zip code, age, and gender)

VeriSign provides three levels, or classes, of security for public-key certificates. A user requests a certificate online at VeriSign's Web site or other participating Web sites. Class 1 and Class 2 requests are processed on line, and in most cases take only a few seconds to approve. Briefly, the following procedures are used:

For Class 1 Digital IDs, VeriSign confirms the user's e-mail address by sending a PIN and Digital ID pick-up information to the e-mail address provided in the application.

For Class 2 Digital IDs, VeriSign verifies the information in the application through an automated comparison with a consumer database in addition to performing all of the checking associated with a Class 1 Digital ID. Finally, confirmation is sent to the specified postal address alerting the user that a Digital ID has been issued in his or her name.

For Class 3 Digital IDs, VeriSign requires a higher level of identity assurance. An individual must prove his or her identity by providing notarized credentials or applying in person.

Enhanced Security Services

As of this writing, three enhanced security services have been proposed in an Internet draft.:

Signed receipts: A signed receipt may be requested in a SignedData object. Returning a signed receipt provides proof of delivery to the originator of a message and allows the originator to demonstrate to a third party that the recipient received the message.

Security labels: A security label may be included in the authenticated attributes of a SignedData object. A security label is a set of security information regarding the sensitivity of the content that is protected by S/MIME encapsulation. The labels may be used for access control, by indicating which users are permitte access to an object.

Secure mailing lists: When a user sends a message to multiple recipients, a certain amount of per-recipient processing is required, including the use of each recipient's public key. The user can be relieved of this work by employing the services of an S/MIME Mail List Agent (MLA). An MLA can take a single incoming message, perform the recipient-specific encryption for each recipient, and forward the message. The originator of a message need only send the message to the MLA, with encryption performed using the MLA's public key.

Key Management

- all cryptographic systems have the problem of how to securely and reliably distribute the keys used
- in many cases, failures in a secure system are due not to breaking the algorithm, but to breaking the key distribution scheme
- ideally the distribution protocol should be formally verified, recent advances make this more achievable
- possible key distribution techniques include:
- physical delivery by secure courier eg code-books used submarines
- one-time pads used by diplomatic missions
- registration name and password for computers

- authentication key server (private key, eg Kerberos) have an on-line server trusted by all clients server has a unique secret key shared with each client server negotiates keys on behalf of clients
- public notary (public key, eg SPX) have an off-line server trusted by all clients
- server has a well known public key
- server signs public key certificates for each client

Authentication Protocols

- if using a key server, must use some protocol between user and server
- this protocol should be validated, formal techniques exist to acheive this (Ban logic provers

Challenge-Response

- basic technique used to ensure a password is never sent in the clear
- given a client and a server share a key
  - server sends a random challenge vector
  - client encrypts it with private key and returns this
  - server verifies response with copy of private key
- can repeat protocol in other direction to authenticate server to client (2-way authentication)
- in simplest form, keys are physically distributed before secure comminications is required
- in more complex forms, keys are stored in a central trusted key server

Needham-Schroeder

- original third-party key distribution protocol

R M Needham, M D Schroeder, "Using Encryption for Authentication in Large Networks of Computers", CACM, 21(12), Dec 1978, pp993-998

- given Alice want to communicate with Bob, and have a Key Server S, protocol is:

Message 1 A -> S A, B, N_a

Message 2 S -> A $E_{Kas}\{N_a , B, K_{ab}, E_{Kbs}\{K_{ab}, A\} \}$

Message 3 A -> B $E_{Kbs}\{K_{ab}, A\}$

Message 4 B -> A $E_{Kab}\{N_b\}$

Message 5 A-> B $E_{Kab}\{N_b\text{-}1\}$

nb: N_a is a random value chosen by Alice, N_b random chosen by Bob

- after this protocol runs, Alice and Bob share a secret session key K_{ab} for secure communication

  - including a timestamp in messages 1 to 3, which requires synchronized clocks (by Denning & Sacco 81)

  - having A ask B for a random value Jb to be sent to S for return in $E_{Kbs}\{Kab, A, Jb\}$ (by Needham & Schroeder 87)

- many other protocols exist but care is needed

KEY MANAGEMENT

- Public-key encryption helps address key distribution problems
- Have two aspects:

o Distribution of public keys

o Use of public-key encryption to distribute secret keys **Distribution of Public Keys**

Distribution of Public Keys can be done in one of the four ways:

Public announcement

Publicly available directory

Public-key authority

Public-key certificates

Public Announcement

- Users distribute public keys to recipients or broadcast to community at large

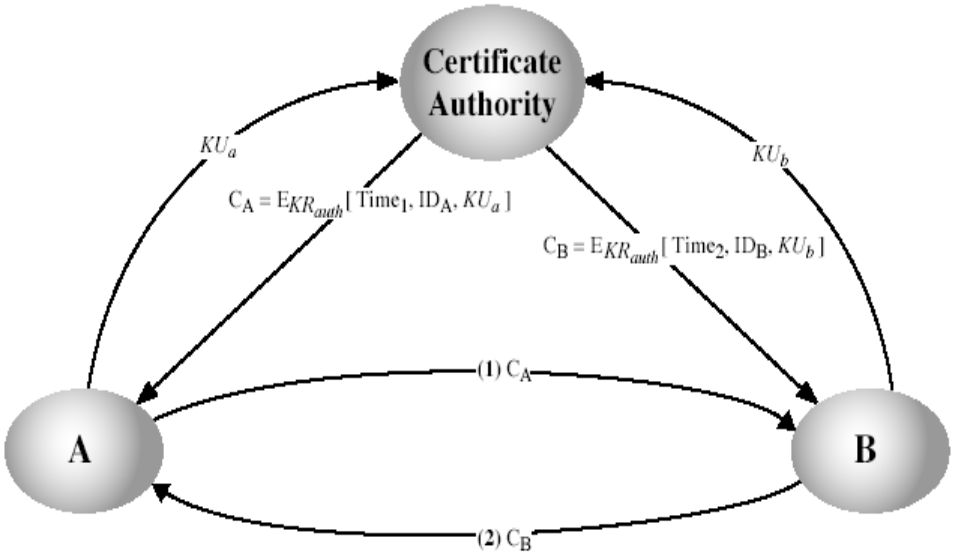
o eg. Append PGP keys to email messages or post to news groups or email list

- Major weakness is forgery

o Anyone can create a key claiming to be someone else and broadcast it

o Until forgery is discovered can masquerade as claimed user

- Certificates allow key exchange without real-time access to public-key authority
- A certificate binds **identity** to **public key**
o Usually with other info such as period of validity, rights of use etc
- With all contents **signed** by a trusted Public-Key or Certificate Authority (CA)
- Can be verified by anyone who knows the public-key authorities public-key



Publicly Available Directory

- Can obtain greater security by registering keys with a public directory

- Directory must be trusted with properties:

o Contains {name, public-key} entries
o Participants register securely with directory
o Participants can replace key at any time
o Directory is periodically published
o Directory can be accessed electronically
- Still vulnerable to tampering or forgery

Public-Key Authority

- Improve security by tightening control over distribution of keys from directory
- Has properties of directory
- Requires users to know public key for the directory

• Users interact with directory to obtain any desired public key securely

o Does require real-time access to directory when keys are needed

Kerberos - An Example of a Key Server

- trusted key server system developed by MIT
- provides centralised third-party authentication in a distributed network
- access control may be provided for each computing resource in either a local or remote network

(realm) has a Key Distribution Centre (KDC), containing a database of:

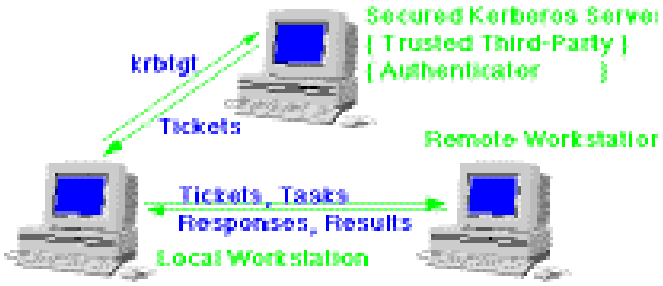
- principles (customers and services)
- encryption keys
- basic third-party authentication scheme
- KDC provides non-corruptible authentication credentials (tickets or tokens)

Kerberos - Initial User Authentication

- user requests an initial ticket from KDC
- used as basis for all remote access requests

Kerberos - Request for a Remote Service

- user requests access to a remote service
  - obtains a ticket from KDC protected with remote key
  - sends ticket with request to remote server



Kerberos - in practise

- currently have two Kerberos versions
  - 4 : restricted to a single realm
  - 5 : allows inter-realm authentication, in beta test

Kerberos v5 is an Internet standard specified in RFC1510, and used by many utilities to use Kerberos need to have a KDC on your network need to have applications running on all participating systems major problem - US export restrictions

Kerberos cannot be directly distributed outside the US in source format (& binary versions must obscure crypto routine entry points and have no encryption) else crypto libraries must be reimplemented locally

X.509 - Directory Authentication Service

- part of CCITT X.500 directory services defines framework for authentication services
- directory may store public-key certificates uses public-key cryptography and digital signatures algorithms not standardized but RSA is recommended

X.509 Certificate

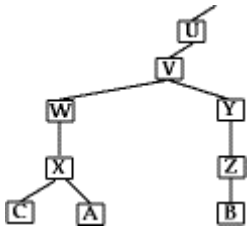
- issued by a Certification Authority (CA) each certificate contains:
  - version
  - serial number (unique within CA)

- algorithm identifier (used to sign certificate)
- issuer (CA)
- period of validity (from - to dates)
- subject (name of owner)
- public-key (algorithm, parameters, key)
- signature (of hash of all fields in certificate)

- any user with access to CA can get any certificate from it
- only the CA can modify a certificate

CA Hierarchy

- CA form a hierarchy
- each CA has certificates for clients and parent
- each client trusts parents certificates
- enable verification of any certificate from one CA by users of all other CAs in hierarchy
- X<<A>> means certificate for A signed by authority X



- A acquires B certificate following chain:
- X<<W>>W<<V>>V<<Y>>Y<<Z>>Z<<B>>
- B acquires A certificate following chain:
- Z<<Y>>Y<<V>>V<<W>>W<<X>>X<<A>>

Authentication Procedures

- X.509 includes three alternative authentication procedures

One-Way Authentication

- 1 message (A->B) to establish
  - identity of A and that messages is from A
  - message intended for B
  - integrity & originality of message

Two-Way Authentication

- 2 messages (A->B, B->A) which also establishes
  - identity of B and that replay is from B
  - reply intended for A
  - integrity & originality of reply

Three-Way Authentication

- 3 messages (A->B, B->A, A->B) which enables
  - above authentication without syncronised clocks

DIFFIE-HELLMAN KEY EXCHANGE

The purpose of the algorithm is to enable two users to exchange a key securely that can then be used for subsequent encryption of messages.

The Diffie-Hellman algorithm depends for its effectiveness on the difficulty of computing discrete logarithms. First, we define a primitive root of a prime number p as one whose power generate all the integers from 1 to (p-1) i.e., if „a" is a primitive root of a prime number p, then the numbers

a mod p, a^2 mod p, … a^{p-1} mod p are distinct and consists of integers from 1 to (p-1) in some permutation. For any integer „b" and a primitive root „a" of a prime number „p", we can find a unique exponent „i" such that

$b \equiv a^i \bmod p$ where $0 \leq i \leq (p-1)$

The exponent „i" is referred to as discrete logarithm. With this background, we can define

Diffie Hellman key exchange as follows:

There are publicly known numbers: a prime number „q" and an integer α that is primitive root of q. suppose users A and B wish to exchange a key. User A selects a random integer $X_A < q$ and computes $Y_A = \alpha^{XA} \bmod q$. Similarly, user B independently selects a random integer $X_B < q$ and computes $Y_B = \alpha^{XB} \bmod q$. Each side keeps the X value private and makes the Y value available publicly to the other side. User A computes the key as

$K = (Y_B)^{XA} \bmod q$ and

User B computes the key as

$K = (Y_A)^{XB} \bmod q$

These two calculations produce identical results.

$K = (Y_B)^{XA} \bmod q$

$= (\alpha^{XB} \bmod q)^{XA} \bmod q$

$= (\alpha^{XB})^{XA} \bmod q$

$= (\alpha^{XA})^{XB} \bmod q$

$= (\alpha^{XA} \bmod q)^{XB} \bmod q$

$= (Y_A)^{XB} \bmod q$

The result is that two sides have exchanged a secret key.

The security of the algorithm lies in the fact that, while it is relatively easy to calculate exponentials modulo a prime, it is very difficult to calculate discrete logarithms. For large primes, the latter task is considered infeasible.

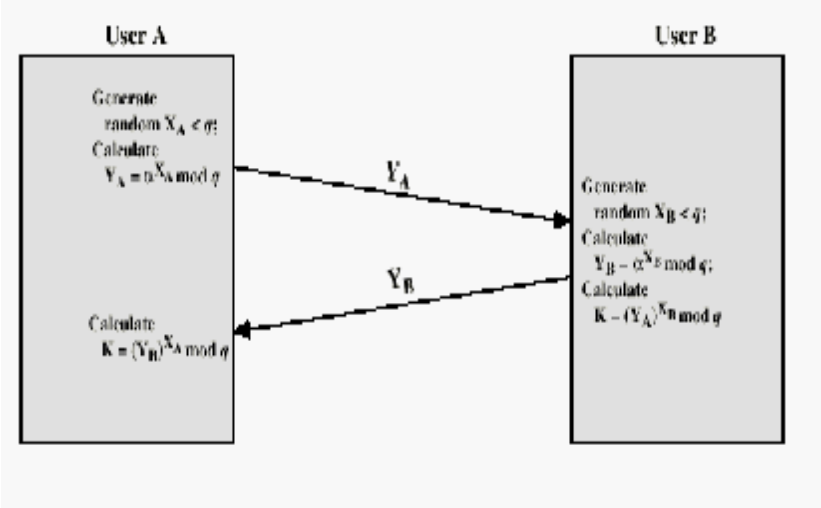


Fig: Diffie Hellman Key exchange

4 Security in Practise - Secure Email

- email is one of the most widely used and regarded network services
- currently message contents are not secure
  - may be inspected either in transit
  - or by suitably priviledged users on destination system
- Email Privacy Enhancement Services
  - confidentiality (protection from disclosure)
  - authentication (of originator)
  - message integrity (protection from modification)
  - non-repudiation of origin
  - (protection from denial by sender)
- can't assume real-time access to a trusted key server
- often implement using Email Encapsulation

PEM

- Privacy Enhanced Mail
- Internet standard for security enhancements to Internet (RFC822) email
  - developed by a Working group of the IETF
  - specified in RFC1421, RFC1422, RFC1423, RFC1424
- uses message encapsulation to add features
- confidentiality - DES encryption in CBC mode
- integrity - DES encrypted MIC (MD2/MD5)
- authentication - DES/RSA encrypted MIC
- non-repudiation - RSA encrypted MIC

PEM - Key Management

- central key server (private-key)
  - requires access to on-line server
- public-key certificates
  - uses X.509 Directory Service Strong Authentication to protect key certificates
  - signed by a Certification Authority (CA)
  - CAs form a hierarchy to permit cross-validation of certificates
  - CAs must be licenced by RSA Data Inc.
  - currently only licensed in US/Canada

PGP

- Pretty Good Privacy
- widely used de facto secure email standard
  - developed by Phil Zimmermann

- available on Unix, PC, Macintosh and Amiga systems
- free!!!!

- confidentiality - IDEA encryption
- integrity - RSA encrypted MIC (MD5)
- authentication & non-repudiation - RSA encrypted MIC
- uses grass-roots key distribution
  - trusted introducers used to validate keys
  - no certification authority hierarchy needed

PGP - In Use

- all PGP functions are performed by a single program
- must be integrated into existing email/news
- each user has a keyring of known keys
  - containing their own public and private keys (protected by a password)
  - public keys given to you directly by a person
  - public keys signed by trusted introducers
- used to sign/encrypt your messages
- used to validate messages received

Sample PGP Message

-----BEGIN PGP SIGNED MESSAGE-----

May all your signals trap
May your references be bounded
All memory aligned
Floats to ints be rounded

Lawrie

-----BEGIN PGP SIGNATURE-----

Version: 2.3

iQBzAgUBLdl1RILpoub8ek7fAQF2nwLuJwVPh8iiFrksXSCe6z37ZdV37pXvsYyz0WAnCBCdpu55

yId5/kVhmvusTo10zUHPssPwB99TQq9YsduSfkVeILjfJNJEuUWQkJl8dWvaB+IIEEodF0Xpbc23krn

uOA==

=hn90

-----END PGP SIGNATURE-----

PGP - Issues

- were questions of legality, but PGP may now be legally used by anyone in the world:
  - noncommercial use in US/Canada with licenced MIT version
  - commercial use in US/Canada with Viacrypt version
  - noncommercial use outside the US is probably legal with (non US sourced) international version
  - commercial use outside the US requires an IDEA licence for the international version
- is on-going legal battle in US over its original export between US govt and Phil Zimmermann

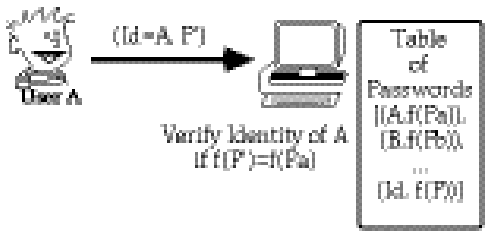
Security in Practice - SNMP

- SNMP is a widely used network management protocol
- comprises
  - management station
  - management agent with
  - its management information base (MIB)
  - linked by network management protocol (GET,SET)
- SNMP v1 lacks any security (GET and SET open if there)
- SNMP v2 includes security extensions for
  - message authentication (keyed MD5)

- message secrecy (DES)
- based on the SNMPv2 **party** (sender & receiver roles)
- used for access control & key management
- all associated information stored in a party MIB
- assumes syncronised clocks (within a set interval)

User Authentication

- user authentication (identity verification)
- convince system of your identity
- before it can act on your behalf
- sometimes also require that the computer verify its identity with the user
- user authentication is based on three methods
- what you know
- what you have
- what you are
- all then involve some validation of information supplied against a table of possible values based on users claimed identity



What you Know

Passwords or Pass-phrases

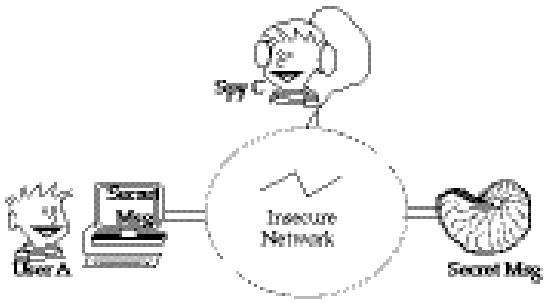
- prompt user for a login name and password
- verify identity by checking that password is correct

- on some (older) systems, password was stored in the clear (this is now regarded as insecure, since breakin compromises all users of the system)
  - more often use a one-way function, whose output cannot easily be used to find the input value either takes a fixed sized input (eg 8 chars)
  - or based on a hash function to accept a variable sized input to create the value
- important that passwords are selected with care to reduce risk of exhaustive search

Denning Computer (In) security Fig 2 & 3, pp111-12

One-shot Passwords

- one problem with traditional passwords is caused by eavesdropping theit transfer over an insecure network



- one possible solution is to use one-shot (one-time) passwords
- these are passwords used once only
- future values cannot be predicted from older values

either generate a printed list, and keep matching list on system to be accessed (cf home banking) or use an algorithm based on a one-way function f (eg MD5) to generate previous values in series (eg SKey)

- start with a secret password s, and number N
  - $p_{(0)} = f^{N}(s)$
- next password in series is
  - $p_{(1)} = f^{N-1}(s)$
- must reset password after N uses

generally good only for infrequent access

INTRUDERS

One of the most publicized attacks to security is the intruder, generally referred to as hacker or cracker. Three classes of intruders are as follows:

- **Masquerader** – an individual who is not authorized to use the computer and who penetrates a system's access controls to exploit a legitimate user's account.

Misfeasor – a legitimate user who accesses data, programs, or resources for which such access is not authorized, or who is authorized for such access but misuse his or her privileges.

Clandestine user – an individual who seizes supervisory control of the system and uses this control to evade auditing and access controls or to suppress audit collection.

The masquerader is likely to be an outsider; the misfeasor generally is an insider; and the clandestine user can be either an outsider or an insider.

Intruder attacks range from the benign to the serious. At the benign end of the scale, there are many people who simply wish to explore internets and see what is out there. At the serious end are individuals who are attempting to read privileged data, perform unauthorized modifications to data, or disrupt the system. Benign intruders might be tolerable, although they do consume resources and may slow performance for legitimate users. However there is no way in advance to know whether an intruder will be benign or malign.

An analysis of previous attack revealed that there were two levels of hackers:

The high levels were sophisticated users with a thorough knowledge of the technology.

The low levels were the 'foot soldiers' who merely use the supplied cracking programs with little understanding of how they work.

one of the results of the growing awareness of the intruder problem has been the establishment of a number of Computer Emergency Response Teams (CERT). these co- operative ventures collect information about system vulnerabilities and disseminate it to systems managers. Unfortunately, hackers can also gain access to CERT reports.

In addition to running password cracking programs, the intruders attempted to modify login software to enable them to capture passwords of users logging onto the systems.

Intrusion techniques

The objective of the intruders is to gain access to a system or to increase the range of privileges

accessible on a system. Generally, this requires the intruders to acquire information that should be protected. In most cases, the information is in the form of a user password.

Typically, a system must maintain a file that associates a password with each authorized user. If such a file is stored with no protection, then it is an easy matter to gain access to it. The password files can be protected in one of the two ways:

One way encryption – the system stores only an encrypted form of user's password. In practice, the system usually performs a one way transformation (not reversible) in which the password is used to generate a key for the encryption function and in which a fixed length output is produced.

Access control – access to the password file is limited to one or a very few accounts.

The following techniques are used for learning passwords.

Try default passwords used with standard accounts that are shipped with the system. Many administrators do not bother to change these defaults.

Exhaustively try all short passwords.

Try words in the system's online dictionary or a list of likely passwords.

Collect information about users such as their full names, the name of their spouse and children, pictures in their office and books in their office that are related to hobbies.

Try user's phone number, social security numbers and room numbers.

Try all legitimate license plate numbers.

Use a torjan horse to bypass restriction on access.

Tap the line between a remote user and the host system. Two principle countermeasures:

Detection – concerned with learning of an attack, either before or after its success.

Prevention – challenging security goal and an uphill bottle at all times.

INTRUSION DETECTION:

Inevitably, the best intrusion prevention system will fail. A system's second line of defense is intrusion detection, and this has been the focus of much research in recent years. This interest is motivated by a number of considerations, including the following:

1. If an intrusion is detected quickly enough, the intruder can be identified and ejected from the system before any damage is done or any data are compromised.
2. An effective intrusion detection system can serve as a deterrent, so acting to prevent intrusions.

3. Intrusion detection enables the collection of information about intrusion techniques that can be used to strengthen the intrusion prevention facility.

Intrusion detection is based on the assumption that the behavior of the intruder differs from that of a legitimate user in ways that can be quantified.

Figure 18.1 suggests, in very abstract terms, the nature of the task confronting the designer of an intrusion detection system. Although the typical behavior of an intruder differs from the typical behavior of an authorized user, there is an overlap in these behaviors. Thus, a loose interpretation of intruder behavior, which will catch more intruders, will also lead to a number of "false positives," or authorized users identified as intruders. On the other hand, an attempt to limit false positives by a tight interpretation of intruder behavior will lead to an increase in false negatives, or intruders not identified as intruders. Thus, there is an element of compromise and art in the practice of intrusion detection.

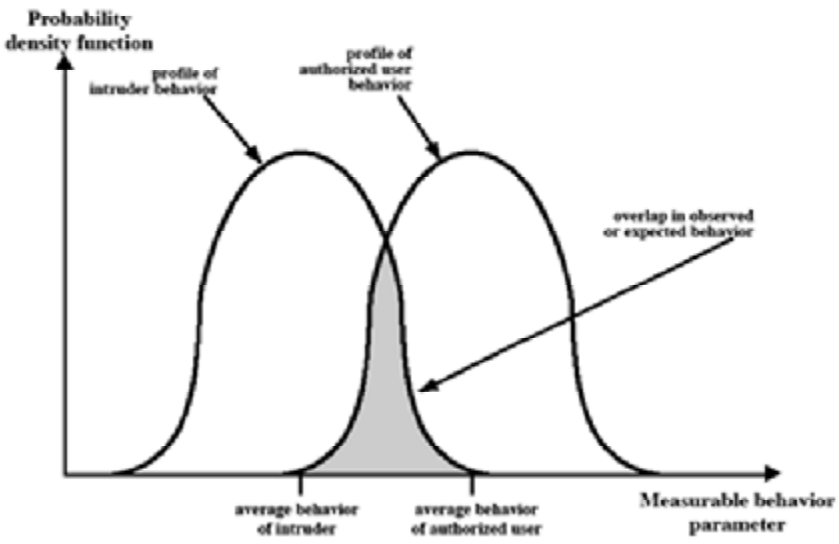


Figure 18.1 Profiles of Behavior of Intruders and Authorized Users

identifies the following approaches to intrusion detection:

1. **Statistical anomaly detection**: Involves the collection of data relating to the behavior of legitimate users over a period of time. Then statistical tests are applied to observed behavior to

determine with a high level of confidence whether that behavior is not legitimate user behavior.

a. **Threshold detection**: This approach involves defining thresholds, independent of user, for the frequency of occurrence of various events.

b. **Profile based:** A profile of the activity of each user is developed and used to detect changes in the behavior of individual accounts.

2. **Rule-based detection**: Involves an attempt to define a set of rules that can be used to decide that a given behavior is that of an intruder.

a. **Anomaly detection**: Rules are developed to detect deviation from previous usage patterns.

b. **Penetration identification**: An expert system approach that searches for suspicious behavior.

In terms of the types of attackers listed earlier, statistical anomaly detection is effective against masqueraders. On the other hand, such techniques may be unable to deal with misfeasors. For such attacks, rule-based approaches may be able to recognize events and sequences that, in context, reveal penetration. In practice, a system may exhibit a combination of both approaches to be effective against a broad range of attacks.

Audit Records

A fundamental tool for intrusion detection is the audit record. Some record of ongoing activity by users must be maintained as input to an intrusion detection system. Basically, two plans are used:

Native audit records: Virtually all multiuser operating systems include accounting software that collects information on user activity. The advantage of using this information is that no additional collection software is needed. The disadvantage is that the native audit records may not contain the needed information or may not contain it in a convenient form.

Detection-specific audit records: A collection facility can be implemented that generates audit records containing only that information required by the intrusion detection system. One advantage of such an approach is that it could be made vendor independent and ported to a variety of systems. The disadvantage is the extra overhead involved in having, in effect, two accounting packages running on a machine.

Each audit record contains the following fields:

Subject: Initiators of actions. A subject is typically a terminal user but might also be a process acting on behalf of users or groups of users.

Object: Receptors of actions. Examples include files, programs, messages, records, terminals, printers, and user- or program-created structures

Resource-Usage: A list of quantitative elements in which each element gives the amount

used of some resource (e.g., number of lines printed or displayed, number of records read or written, processor time, I/O units used, session elapsed time).

Time-Stamp: Unique time-and-date stamp identifying when the action took place.

Most user operations are made up of a number of elementary actions. For example, a file copy involves the execution of the user command, which includes doing access validation and setting up the copy, plus the read from one file, plus the write to another file. Consider the command

COPY GAME.EXE TO <Library>GAME.EXE

issued by Smith to copy an executable file GAME from the current directory to the <Library> directory. The following audit records may be generated:

Smith	execute	<Library>COPY.EXE	0	CPU = 00002	11058721678

Smith	read	<Smith>GAME.EXE	0	RECORDS = 0	11058721679

Smith	execute	<Library>COPY.EXE	write-viol	RECORDS = 0	11058721680

In this case, the copy is aborted because Smith does not have write permission to <Library>. The decomposition of a user operation into elementary actions has three advantages:

1. Because objects are the protectable entities in a system, the use of elementary actions enables an audit of all behavior affecting an object. Thus, the system can detect attempted subversions of access
2. Single-object, single-action audit records simplify the model and the implementation.
3. Because of the simple, uniform structure of the detection-specific audit records, it may be relatively easy to obtain this information or at least part of it by a straightforward mapping from existing native audit records to the detection-specific audit records.

Statistical Anomaly Detection:

As was mentioned, statistical anomaly detection techniques fall into two broad categories: threshold detection and profile-based systems. **Threshold detection involves** counting the number of occurrences of a specific event type over an interval of time. If the count surpasses what is considered a reasonable number that one might expect to occur, then intrusion is assumed.

Threshold analysis, by itself, is a crude and ineffective detector of even moderately sophisticated attacks. Both the threshold and the time interval must be determined.

Profile-based anomaly detection focuses on characterizing the past behavior of individual users or related groups of users and then detecting significant deviations. A profile may consist of a set of parameters, so that deviation on just a single parameter may not be sufficient in itself to signal an alert.

The foundation of this approach is an analysis of audit records. The audit records provide input to the intrusion detection function in two ways. First, the designer must decide on a number of quantitative metrics that can be used to measure user behavior. Examples of metrics that are useful for profile-based intrusion detection are the following:

Counter: A nonnegative integer that may be incremented but not decremented until it is reset by management action. Typically, a count of certain event types is kept over a particular period of time. Examples include the number of logins by a single user during an hour, the number of times a given command is executed during a single user session, and the number of password failures during a minute.

Gauge: A nonnegative integer that may be incremented or decremented. Typically, a gauge is used to measure the current value of some entity. Examples include the number of logical connections assigned to a user application and the number of outgoing messages queued for a user process.

Interval timer: The length of time between two related events. An example is the length of time between successive logins to an account.

Resource utilization: Quantity of resources consumed during a specified period.

Examples include the number of pages printed during a user session and total time consumed by a program execution.Given these general metrics, various tests can be performed to determine whether current activity fits within acceptable limits. [DENN87] lists the following approaches that may be taken:

Mean and standard deviation

Multivariate

Markov process

Time series

Operational

The simplest statistical test is to measure the mean and standard deviation of a parameter over some historical period. This gives a reflection of the average behavior and its variability.

A multivariate model is based on correlations between two or more variables. Intruder behavior may be characterized with greater confidence by considering such correlations (for example, processor time and resource usage, or login frequency and session elapsed time).

A Markov process model is used to establish transition probabilities among various states. As an example, this model might be used to look at transitions between certain commands. A time series model focuses on time intervals, looking for sequences of events that happen too rapidly or too slowly. A variety of statistical tests can be applied to characterize abnormal timing. Finally, an operational model is based on a judgment of what is considered abnormal, rather than an automated analysis of past audit records. Typically, fixed limits are defined and intrusion is suspected for an observation that is outside the limits.

Rule-Based Intrusion Detection

Rule-based techniques detect intrusion by observing events in the system and applying a set of rules that lead to a decision regarding whether a given pattern of activity is or is not suspicious. **Rule-based anomaly detection** is similar in terms of its approach and strengths to statistical anomaly detection. With the rule-based approach, historical audit records are analyzed to identify usage patterns and to generate automatically rules that describe those patterns. Rules may represent past behavior patterns of users, programs, privileges, time slots, terminals, and so on. Current behavior is then observed, and each transaction is matched against the set of rules to determine if it conforms to any historically observed pattern of behavior.

As with statistical anomaly detection, rule-based anomaly detection does not require knowledge of security vulnerabilities within the system. Rather, the scheme is based on observing past behavior and, in effect, assuming that the future will be like the past

Rule-based penetration identification takes a very different approach to intrusion detection, one based on expert system technology. The key feature of such systems is the use of rules for identifying known penetrations or penetrations that would exploit known weaknesses.

Example heuristics are the following:

1. Users should not read files in other users' personal directories.
2. Users must not write other users' files.
3. Users who log in after hours often access the same files they used earlier.
4. Users do not generally open disk devices directly but rely on higher-level operating system utilities.
5. Users should not be logged in more than once to the same system.
6. Users do not make copies of system programs.

The Base-Rate Fallacy

To be of practical use, an intrusion detection system should detect a substantial percentage of intrusions while keeping the false alarm rate at an acceptable level. If only a modest percentage of actual intrusions are detected, the system provides a false sense of security. On the other hand, if the system frequently triggers an alert when there is no intrusion (a false alarm), then either system managers will begin to ignore the alarms, or much time will be wasted analyzing the false alarms.

Unfortunately, because of the nature of the probabilities involved, it is very difficult to meet the standard of high rate of detections with a low rate of false alarms. In general, if the actual numbers of intrusions is low compared to the number of legitimate uses of a system, then the false alarm rate will be high unless the test is extremely discriminating.

Distributed Intrusion Detection

Until recently, work on intrusion detection systems focused on single-system stand-alone facilities. The typical organization, however, needs to defend a distributed collection of hosts supported by a LAN Porras points out the following major issues in the design of a distributed intrusion detection system

A distributed intrusion detection system may need to deal with different audit record formats. In a heterogeneous environment, different systems will employ different native audit collection systems and, if using intrusion detection, may employ different formats for security-related audit records.

One or more nodes in the network will serve as collection and analysis points for the data from the systems on the network. Thus, either raw audit data or summary data must be transmitted across the network. Therefore, there is a requirement to assure the integrity and confidentiality of these data.

Either a centralized or decentralized architecture can be used.

Host agent module: An audit collection module operating as a background process on a monitored system. Its purpose is to collect data on security-related events on the host and transmit these

to the central manager.

LAN monitor agent module: Operates in the same fashion as a host agent module except that it analyzes LAN traffic and reports the results to the central manager.

Central manager module: Receives reports from LAN monitor and host agents and processes and correlates these reports to detect intrusion.

A filter is applied that retains only those records that are of security interest.

These records are then reformatted into a standardized format referred to as the host audit record (HAR).

Next, a template-driven logic module analyzes the records for suspicious activity.

At the lowest level, the agent scans for notable events that are of interest independent of any past events.

Examples include failed file accesses, accessing system files, and changing a file's access control.

At the next higher level, the agent looks for sequences of events, such as known attack patterns (signatures).

Finally, the agent looks for anomalous behavior of an individual user based on a historical profile of that user, such as number of programs executed, number of files accessed, and the like.

When suspicious activity is detected, an alert is sent to the central manager.

The central manager includes an expert system that can draw inferences from received data.

The manager may also query individual systems for copies of HARs to correlate with those from other agents.

The LAN monitor agent also supplies information to the central manager.

The LAN monitor agent audits host-host connections, services used, and volume of traffic.

It searches for significant events, such as sudden changes in network load, the use of security-related services, and network activities such as rlogin.

The architecture depicted in Figures 18.2 and 18.3 is quite general and flexible. It offers a foundation for a machine-independent approach that can expand from stand-alone intrusion detection to a system that is able to correlate activity from a number of sites and networks to detect suspicious activity that would otherwise remain undetected.

Honeypots

A relatively recent innovation in intrusion detection technology is the honeypot. Honeypots are decoy systems that are designed to lure a potential attacker away from critical systems. Honeypots are designed to

divert an attacker from accessing critical systems

collect information about the attacker's activity

encourage the attacker to stay on the system long enough for administrators to respond

These systems are filled with fabricated information designed to appear valuable but that a legitimate user of the system wouldn't access. Thus, any access to the honeypot is suspect.

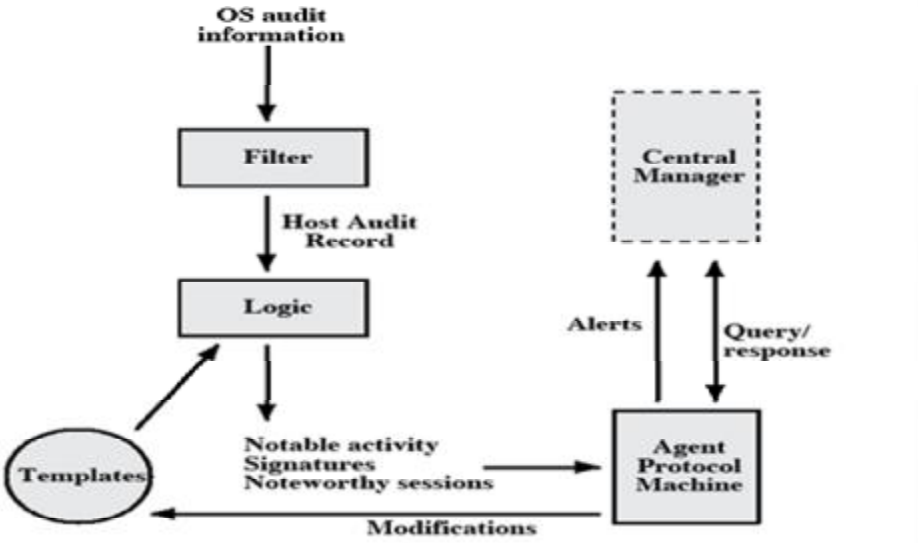


Figure 18.3 Agent Architecture

Intrusion Detection Exchange Format

To facilitate the development of distributed intrusion detection systems that can function across a wide range of platforms and environments, standards are needed to support interoperability. Such standards are the focus of the IETF Intrusion Detection Working Group. The outputs of this working group include the following:

1. A requirements document, which describes the high-level functional requirements for communication between intrusion detection systems and with management systems, including the rationale for those requirements.
2. A common intrusion language specification, which describes data formats that satisfy the requirements.
3. A framework document, which identifies existing protocols best used for communication between intrusion detection systems, and describes how the devised data formats relate to them.

FIREWALLS

Firewall design principles

Internet connectivity is no longer an option for most organizations. However, while internet access provides benefits to the organization, it enables the outside world to reach and interact with local network assets. This creates the threat to the organization. While it is possible to equip each workstation and server on the premises network with strong security features, such as intrusion protection, this is not a practical approach. The alternative, increasingly accepted, is the firewall.

The firewall is inserted between the premise network and internet to establish a controlled link and to erect an outer security wall or perimeter. The aim of this perimeter is to protect the premises network from internet based attacks and to provide a single choke point where security and audit can be imposed. The firewall can be a single computer system or a set of two or more systems that cooperate to perform the firewall function.

Firewall characteristics:

All traffic from inside to outside, and vice versa, must pass through the firewall.

This is achieved by physically blocking all access to the local network except via the firewall. Various configurations are possible.

only authorized traffic, as defined by the local security policy, will be allowed to pass.

Various types of firewalls are used, which implement various types of security policies.

the firewall itself is immune to penetration. This implies that use of a trusted system with a secure operating system. This implies that use of a trusted system with a secure operating system.

Four techniques that firewall use to control access and enforce the site's security policy is as follows:

Service control – determines the type of internet services that can be accessed, inbound or outbound. The firewall may filter traffic on this basis of IP address and TCP port number; may provide proxy software that receives and interprets each service request before passing it on; or may host the server software itself, such as web or mail service.

Direction control – determines the direction in which particular service request may be initiated and allowed to flow through the firewall.

User control – controls access to a service according to which user is attempting to access it.

Behavior control – controls how particular services are used.

Capabilities of firewall

A firewall defines a single choke point that keeps unauthorized users out of the protected network, prohibits potentially vulnerable services from entering or leaving the network, and provides protection from various kinds of IP spoofing and routing attacks.

A firewall provides a location for monitoring security related events. Audits and alarms can be implemented on the firewall system.

A firewall is a convenient platform for several internet functions that are not security related.

A firewall can serve as the platform for IPsec.

Limitations of firewall

The firewall cannot protect against attacks that bypass the firewall. Internal systems may have dial-out capability to connect to an ISP. An internal LAN may support a modem pool that provides dial-in capability for traveling employees and telecommuters.

the firewall does not protect against internal threats. The firewall does not protect against internal threats, such as a disgruntled employee or an employee who unwittingly cooperates with an external attacker.

The firewall cannot protect against the transfer of virus-infected programs or files.

Because of the variety of operating systems and applications supported inside the perimeter, it would be impractical and perhaps impossible for the firewall to scan all incoming files, e-mail, and

messages for viruses.

Types of firewalls

There are 3 common types of firewalls.

Packet filters

Application-level gateways

Circuit-level gateways

Packet filtering router

A packet filtering router applies a set of rules to each incoming IP packet and then forwards or discards the packet. The router is typically configured to filter packets going in both directions. Filtering rules are based on the information contained in a network packet:

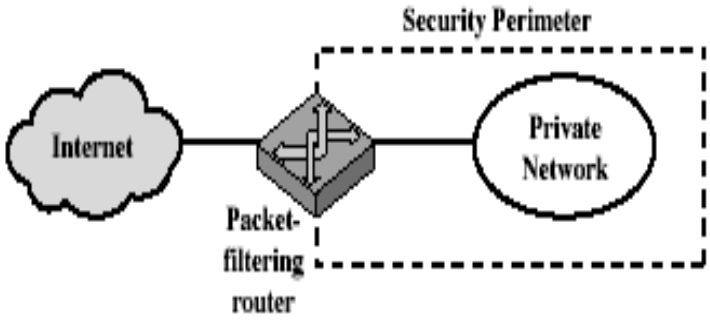
Source IP address – IP address of the system that originated the IP packet.

Destination IP address – IP address of the system, the IP is trying to reach.

Source and destination transport level address – transport level port number.

IP protocol field – defines the transport protocol.

Interface – for a router with three or more ports, which interface of the router the packet come from or which interface of the router the packet is destined for.



(a) Packet-filtering router

The packet filter is typically set up as a list of rules based on matches to fields in the IP or TCP header. If there is a match to one of the rules, that rule is invoked to determine whether to forward or discard the packet. If there is no match to any rule, then a default action is taken.

Two default policies are possible:

Default = discard: That which is not expressly permitted is prohibited.

Default = forward: That which is not expressly prohibited is permitted.

The default discard policy is the more conservative. Initially everything is blocked, and services must be added on a case-by-case basis. This policy is more visible to users, who are most likely to see the firewall as a hindrance. The default forward policy increases ease of use for end users but provides reduced security.

Advantages of packet filter router

Simple

Transparent to users

Very fast

Weakness of packet filter firewalls

Because packet filter firewalls do not examine upper-layer data, they cannot prevent attacks that employ application specific vulnerabilities or functions.

Because of the limited information available to the firewall, the logging functionality present in packet filter firewall is limited.

It does not support advanced user authentication schemes.

They are generally vulnerable to attacks such as layer address spoofing.

Some of the attacks that can be made on packet filtering routers and the appropriate counter measures are the following:

IP address spoofing –

the intruders transmit packets from the outside with a source IP address field containing an address of an internal host.

Countermeasure: to discard packet with an inside source address if the packet arrives on an external interface.

Source routing attacks – the source station specifies the route that a packet should take as it crosses the internet; i.e., it will bypass the firewall.

Countermeasure: to discard all packets that uses this option.

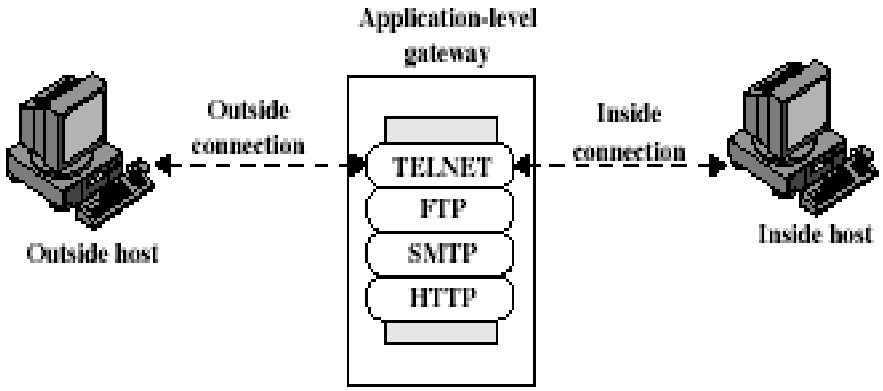
Tiny fragment attacks – the intruder create extremely small fragments and force the TCP header information into a separate packet fragment. The attacker hopes that only the first fragment is examined and the remaining fragments are passed through.

Countermeasure: to discard all packets where the protocol type is TCP and the IP Fragment offset is equal to 1.

Application level gateway

An Application level gateway, also called a proxy server, acts as a relay of application level traffic. The user contacts the gateway using a TCP/IP application, such as Telnet or FTP, and the gateway asks the user for the name of the remote host to be accessed. When the user responds and provides a valid user ID and authentication information, the gateway contacts the application on the remote host and relays TCP segments containing the application data between the two endpoints.

Application level gateways tend to be more secure than packet filters. It is easy to log and audit all incoming traffic at the application level. A prime disadvantage is the additional processing overhead on each connection.

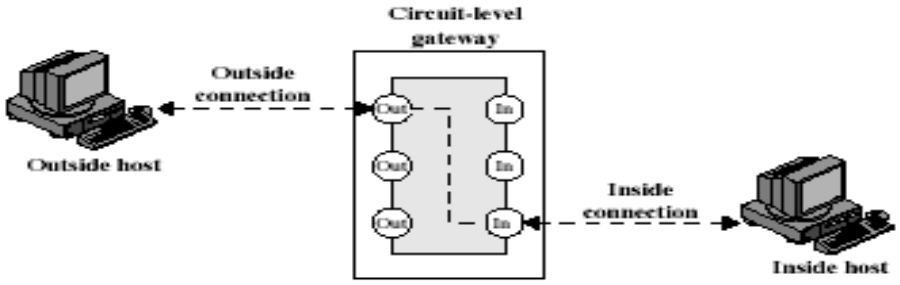


(b) Application-level gateway

Circuit level gateway

Circuit level gateway can be a stand-alone system or it can be a specified function performed by an application level gateway for certain applications. A Circuit level gateway does not permit an end-to-end TCP connection; rather, the gateway sets up two TCP connections, one between itself and a TCP user on an inner host and one between itself and a TCP user on an outer host. Once the two connections are established, the gateway typically relays TCP segments from one connection to the other without examining the contents. The security function consists of determining which connections will be allowed.

A typical use of Circuit level gateways is a situation in which the system administrator trusts the internal users. The gateway can be configured to support application level or proxy service on inbound connections and circuit level functions for outbound connections.



(c) Circuit-level gateway

Basiton host

It is a system identified by the firewall administrator as a critical strong point in the network's security. The Bastion host serves as a platform for an application level and circuit level gateway. Common characteristics of a Basiton host are as follows:

The Bastion host hardware platform executes a secure version of its operating system, making it a trusted system.

Only the services that the network administrator considers essential are installed on the Bastion host.

It may require additional authentication before a user is allowed access to the proxy services.

Each proxy is configured to support only a subset of standard application's command set.

Each proxy is configured to allow access only to specific host systems.

Each proxy maintains detailed audit information by logging all traffic, each connection and the duration of each connection.

Each proxy is independent of other proxies on the Bastion host.

A proxy generally performs no disk access other than to read its initial configuration file.

Each proxy runs on a non-privileged user in a private and secured directory on the Bastion host.

Firewall configurations

There are 3 common firewall configurations.

1. Screened host firewall, single-homed basiton configuration

In this configuration, the firewall consists of two systems: a packet filtering router and a bastion host. Typically, the router is configured so that

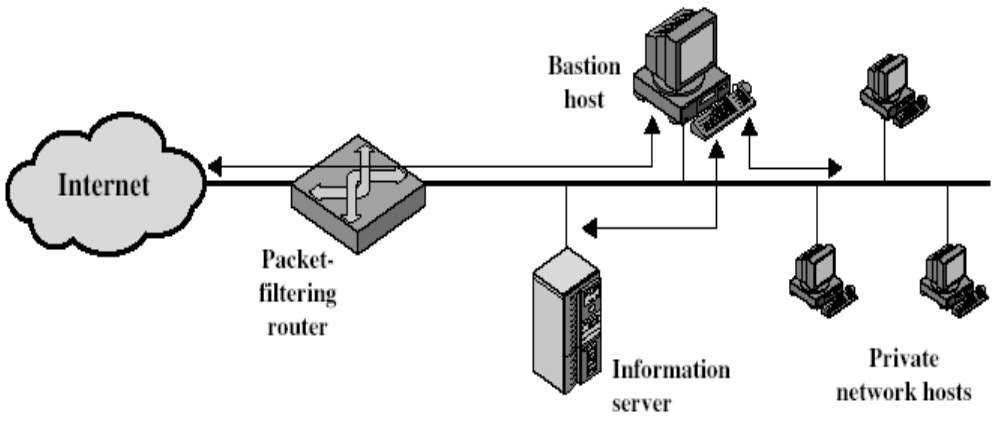
For traffic from the internet, only IP packets destined for the basiton host are allowed in.

For traffic from the internal network, only IP packets from the basiton host are allowed out.

The basiton host performs authentication and proxy functions. This configuration has greater security than simply a packet filtering router or an application level gateway alone, for two reasons:

This configuration implements both packet level and application level filtering, allowing for considerable flexibility in defining security policy.

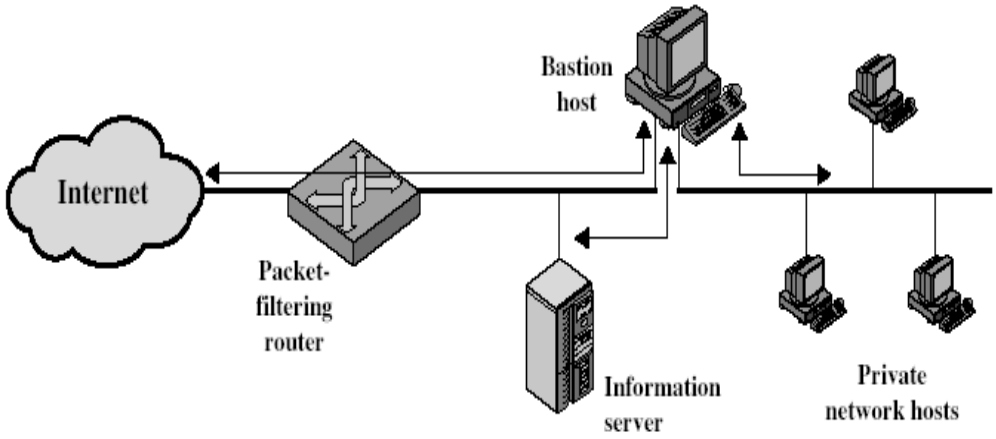
An intruder must generally penetrate two separate systems before the security of the internal network is compromised.



(a) Screened host firewall system (single-homed bastion host)

2. Screened host firewall, dual homed basiton configuration

In the previous configuration, if the packet filtering router is compromised, traffic could flow directly through the router between the internet and the other hosts on the private network. This configuration physically prevents such a security break.



(b) Screened host firewall system (dual-homed bastion host)

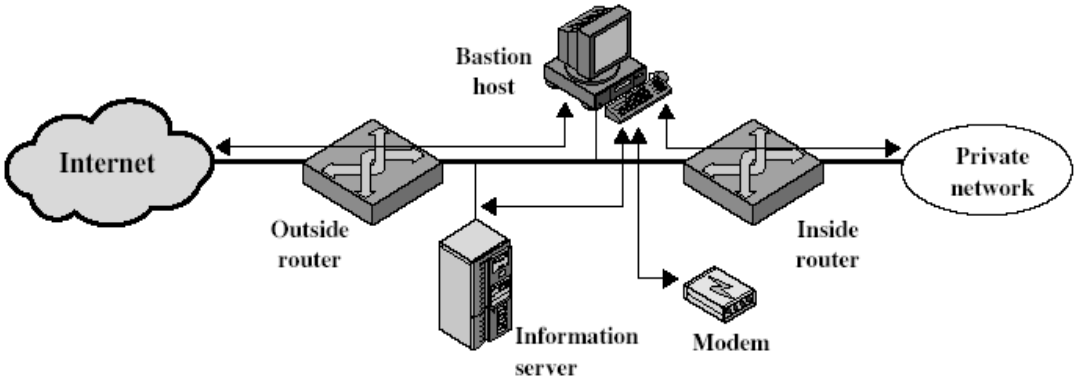
3. Screened subnet firewall configuration

In this configuration, two packet filtering routers are used, one between the basiton host and internet and one between the basiton host and the internal network. This configuration creates an isolated subnetwork, which may consist of simply the basiton host but may also include one or more information servers and modems for dial-in capability. Typically both the internet and the internal network have access to hosts on the screened subnet, but traffic across the screened subnet is blocked. This configuration offers several advantages:

There are now three levels of defense to thwart intruders.

The outside router advertises only the existence of the screened subnet to the internet; therefore the internal network is invisible to the internet.

Similarly, the inside router advertises only the existence of the screened subnet to the internal network; therefore the systems on the internal network cannot construct direct routes to the internet.



(c) Screened-subnet firewall system

Trusted systems

One way to enhance the ability of a system to defend against intruders and malicious programs is to implement trusted system technology.

Data access control

Following successful logon, the user has been granted access to one or set of hosts and applications. This is generally not sufficient for a system that includes sensitive data in its database. Through the user access control procedure, a user can be identified to the system. Associated with each user, there can be a profile that specifies permissible operations and file accesses. The operating system can then enforce rules based on the user profile. The database management system, however, must control access to specific records or even portions of records. The operating system may grant a user permission to access a file or use an application, following which there are no further security checks, the database management system must make a decision on each individual access attempt. That decision will depend not only on the user's identity but also on the specific parts of the data being accessed and even on the information already divulged to the user.

A general model of access control as exercised by an file or database management system is that of an access matrix. The basic elements of the model are as follows:

Subject: An entity capable of accessing objects. Generally, the concept of subject equates with that of process.

Object: Anything to which access is controlled. Examples include files, portion of files, programs, and segments of memory.

Access right: The way in which the object is accessed by a subject. Examples are read, write and execute.

One axis of the matrix consists of identified subjects that may attempt data access. Typically, this list will consist of individual users or user groups. The other axis lists the objects that may be accessed. Objects may be individual data fields. Each entry in the matrix indicates the access rights of that subject for that object. The matrix may be decomposed by columns, yielding **access control lists.** Thus, for each object, an access control list lists users and their permitted access rights. The access control list may contain a default, or public, entry.

a. Access matrix

Access control list for Program1:

Process1 (Read, Execute)

Access control list for Segment A:

Process1 (Read, Write)

Access control list for Segment B:

Process2 (Read)

b. Access control list

Capability list for Process1: Program1 (Read, Execute) Segment A (Read)

Capability list for Process2:

Segment B (Read)

c. Capability list

Decomposition by rows yields **capability tickets**. A capability ticket specifies authorized objects and operations for a user. Each user has a number of tickets and may be authorized to loan or give them to others. Because tickets may be dispersed around the system, they present a greater security problem than access control lists. In particular, the ticket must be unforgeable. One way to accomplish this is to have the operating system hold all tickets on behalf of users. These tickets would have to be held in a region of memory inaccessible to users.

The concept of Trusted Systems

When multiple categories or levels of data are defined, the requirement is referred to as multilevel security. The general statement of the requirement for multilevel security is that a subject at a high level may not convey information to a subject at a lower or noncomparable level unless that flow accurately reflects the will of an authorized user. For implementation purposes, this requirement is in two parts and is simply stated. A multilevel secure system must enforce:

No read up: A subject can only read an object of less or equal security level. This is

referred to as **simple security property.**

No write down: A subject can only write into an object of greater or equal security level. This is referred to as ***-property (star property).**

These two rules, if properly enforced, provide multilevel security.

Reference Monitor concept

The reference monitor is a controlling element in the hardware and operating system of a computer that regulates the access of subjects to objects on the basis of security parameters of the subject and object. The reference monitor has access to a file, known as the security kernel database that lists the access privileges (security clearance) of each subject and the protection attributes (classification level) of each object. The reference monitor enforces the security rules and has the following properties:

Complete mediation: The security rules are enforced on every access, not just, fr example, when a file is opened.

Isolation: The reference monitor and database are protected from unauthorised modification.

Verifiability: The reference monitor's correctness must be provable. That is, it must be possible to demonstrate mathematically that the reference monitor enforces the security rules and provides complete mediation and isolation. Important security events, such as detected security violations and authorized changes to the security kernel database, are stored in the audit file.

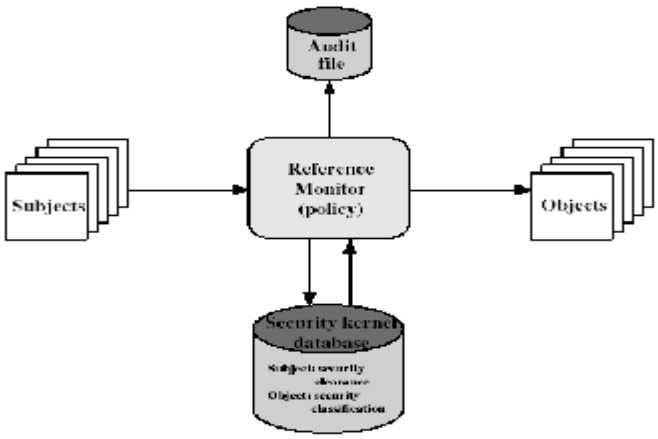


Fig: Reference Monitor Concept

VIRUSES AND RELATED THREATS

Perhaps the most sophisticated types of threats to computer systems are presented by programs that exploit vulnerabilities in computing systems.

Malicious Programs

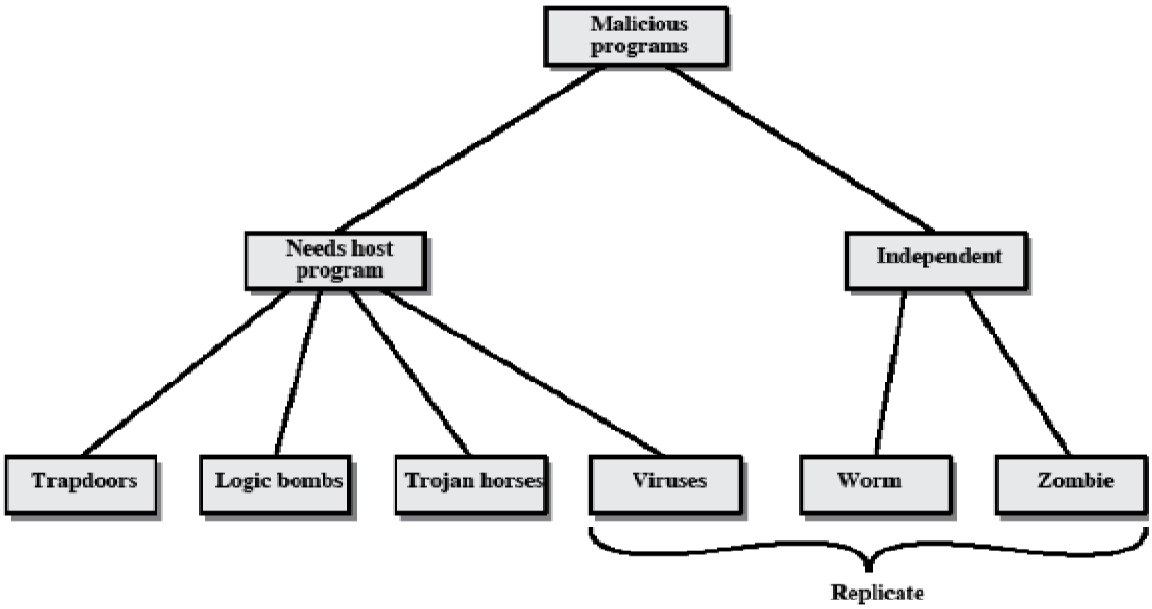


Figure 19.1 Taxonomy of Malicious Programs

Name	Description
Virus	Attaches itself to a program and propagates copies of itself to other programs
Worm	Program that propagates copies of itself to other computers
Logic bomb	Triggers action when condition occurs

Trojan horse	Program that contains unexpected additional functionality
Backdoor	Program modification that allows unauthorized access to functionality
Exploits	Code specific to a single vulnerability or set of vulnerabilities
Downloaders	Program that installs other items on a machine that is under attack Usually, a downloader is sent in an e-mail.
Auto-rooter	Malicious hacker tools used to break into new machines remotely
Kit (virus generator)	Set of tools for generating new viruses automatically
Spammer programs	Used to send large volumes of unwanted e-mail
Flooders	Used to attack networked computer systems with a large volume of traffic to carry out a denial of service (DoS) attack
Keyloggers	Captures keystrokes on a compromised system
Rootkit	Set of hacker tools used after attacker has broken into a computer system and gained root-level access
Zombie	Program activated on an infected machine that is activated to launch attacks on other machines

Malicious software can be divided into two categories: those that need a host program, and those that are independent.

The former are essentially fragments of programs that cannot exist independently of some actual application program, utility, or system program. Viruses, logic bombs, and backdoors are examples. The latter are self-contained programs that can be scheduled and run by the operating system. Worms and zombie programs are examples.

The Nature of Viruses

A virus is a piece of software that can "infect" other programs by modifying them; the modification includes a copy of the virus program, which can then go on to infect other programs.

A virus can do anything that other programs do. The only difference is that it attaches itself to another program and executes secretly when the host program is run. Once a virus is executing, it can perform any function, such as erasing files and programs.

During its lifetime, a typical virus goes through the following four phases:

Dormant phase: The virus is idle. The virus will eventually be activated by some event, such as a date, the presence of another program or file, or the capacity of the disk exceeding some limit. Not all viruses have this stage.

Propagation phase: The virus places an identical copy of itself into other programs or into certain system areas on the disk. Each infected program will now contain a clone of the virus, which will itself enter a propagation phase.

Triggering phase: The virus is activated to perform the function for which it was intended. As with the dormant phase, the triggering phase can be caused by a variety of system events, including a count of the number of times that this copy of the virus has made copies of itself.

Execution phase: The function is performed. The function may be harmless, such as a message on the screen, or damaging, such as the destruction of programs and data files.

Virus Structure

A virus can be prepended or postpended to an executable program, or it can be embedded in some other fashion. The key to its operation is that the infected program, when invoked, will first execute the virus code and then execute the original code of the program.

An infected program begins with the virus code and works as follows.

The first line of code is a jump to the main virus program. The second line is a special marker that is used by the virus to determine whether or not a potential victim program has already been infected with this virus.

When the program is invoked, control is immediately transferred to the main virus program. The virus program first seeks out uninfected executable files and infects them. Next, the virus may perform some action, usually detrimental to the system.

This action could be performed every time the program is invoked, or it could be a logic bomb that triggers only under certain conditions.

Finally, the virus transfers control to the original program. If the infection phase of the program is reasonably rapid, a user is unlikely to notice any difference between the execution of an infected and uninfected program.

A virus such as the one just described is easily detected because an infected version of a program is longer than the corresponding uninfected one. A way to thwart such a simple means of detecting a virus is to compress the executable file so that both the infected and uninfected versions are of identical length.. The key lines in this virus are numbered, and Figure 19.3 [COHE94] illustrates the operation. We assume that program P_1 is infected with the virus CV. When this program is invoked, control passes to its virus, which performs the following steps:

1. For each uninfected file P_2 that is found, the virus first compresses that file to produce P'_2, which is shorter than the original program by the size of the virus.

2. A copy of the virus is prepended to the compressed program.

3. The compressed version of the original infected program, P'_1, is uncompressed.

4. The uncompressed original program is executed.

In this example, the virus does nothing other than propagate. As in the previous example, the virus may include a logic bomb.

Initial Infection

Once a virus has gained entry to a system by infecting a single program, it is in a position to infect some or all other executable files on that system when the infected program executes. Thus, viral infection can be completely prevented by preventing the virus from gaining entry in the first place. Unfortunately, prevention is extraordinarily difficult because a virus can be part of any program outside a system. Thus, unless one is content to take an absolutely bare piece of

Iron and write all one's own system and application programs, one is vulnerable.

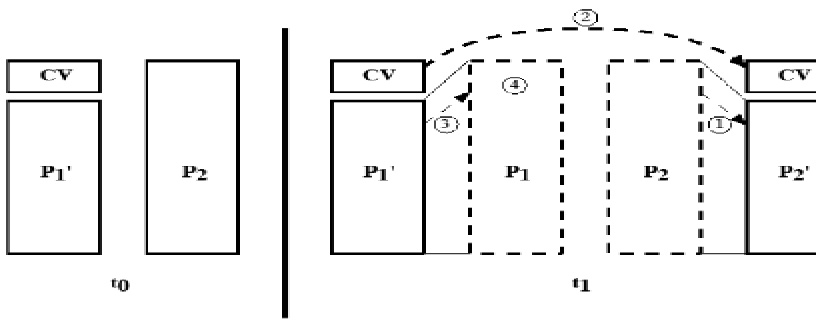


Figure 19.4 A Compression Virus

Types of Viruses

Following categories as being among the most significant types of viruses:

Parasitic virus: The traditional and still most common form of virus. A parasitic virus attaches itself to executable files and replicates, when the infected program is executed, by finding other executable files to infect.

Memory-resident virus: Lodges in main memory as part of a resident system program.

From that point on, the virus infects every program that executes.

Boot sector virus: Infects a master boot record or boot record and spreads when a system is booted from the disk containing the virus.

Stealth virus: A form of virus explicitly designed to hide itself from detection by antivirus software.

Polymorphic virus: A virus that mutates with every infection, making detection by the

"signature" of the virus impossible.

Metamorphic virus: As with a polymorphic virus, a metamorphic virus mutates with every infection. The difference is that a metamorphic virus rewrites itself completely at each iteration, increasing the difficulty of detection. Metamorphic viruses my change their behavior as well as their appearance.

One example of a **stealth virus** was discussed earlier: a virus that uses compression so that the infected program is exactly the same length as an uninfected version. Far more sophisticated techniques are possible. For example, a virus can place intercept logic in disk I/O routines, so that when there is an attempt to read suspected portions of the disk using these routines, the virus will present back the original, uninfected program.

A **polymorphic virus** creates copies during replication that are functionally equivalent but have distinctly different bit patterns.

Macro Viruses

In the mid-1990s, macro viruses became by far the most prevalent type of virus. Macro viruses are particularly threatening for a number of reasons:

1. A macro virus is platform independent. Virtually all of the macro viruses infect Microsoft Word documents. Any hardware platform and operating system that supports Word can be infected.
2. Macro viruses infect documents, not executable portions of code. Most of the information introduced onto a computer system is in the form of a document rather than a program.
3. Macro viruses are easily spread. A very common method is by electronic mail.

Macro viruses take advantage of a feature found in Word and other office applications such as Microsoft Excel, namely the macro. In essence, a macro is an executable program embedded in a word processing document or other type of file. Typically, users employ macros to automate repetitive tasks and thereby save keystrokes. The macro language is usually some form of the Basic programming language. A user might define a sequence of keystrokes in a macro and set it up so that the macro is invoked when a function key or special short combination of keys is input.

Successive releases of Word provide increased protection against macro viruses. For example, Microsoft offers an optional Macro Virus Protection tool that detects suspicious Word files and alerts the customer to the potential risk of opening a file with macros. Various antivirus product vendors have also developed tools to detect and correct macro viruses.

E-mail Viruses

A more recent development in malicious software is the e-mail virus. The first rapidly spreading e-mail viruses, such as Melissa, made use of a Microsoft Word macro embedded in an attachment. If the recipient opens the e-mail attachment, the Word macro is activated. Then

1. The e-mail virus sends itself to everyone on the mailing list in the user's e-mail package.

2. The virus does local damage.

Worms

A worm is a program that can replicate itself and send copies from computer to computer across network connections. Upon arrival, the worm may be activated to replicate and propagate again.

Network worm programs use network connections to spread from system to system. Once active within a system, a network worm can behave as a computer virus or bacteria, or it could implant Trojan horse programs or perform any number of disruptive or destructive actions.

To replicate itself, a network worm uses some sort of network vehicle. Examples include the following:

Electronic mail facility: A worm mails a copy of itself to other systems.

Remote execution capability: A worm executes a copy of itself on another system.

Remote login capability: A worm logs onto a remote system as a user and then uses commands to copy itself from one system to the other.

The new copy of the worm program is then run on the remote system where, in addition to any functions that it performs at that system, it continues to spread in the same fashion.

A network worm exhibits the same characteristics as a computer virus: a dormant phase, a propagation phase, a triggering phase, and an execution phase. The propagation phase

generally performs the following functions:

1. Search for other systems to infect by examining host tables or similar repositories of remote system addresses.

2. Establish a connection with a remote system.

3. Copy itself to the remote system and cause the copy to be run.

As with viruses, network worms are difficult to counter.

The Morris Worm

The Morris worm was designed to spread on UNIX systems and used a number of different techniques for propagation.

1. It attempted to log on to a remote host as a legitimate user. In this method, the worm first attempted to crack the local password file, and then used the discovered passwords and corresponding user IDs. The assumption was that many users would use the same password on different systems. To obtain the passwords, the worm ran a password- cracking program that tried

a. Each user's account name and simple permutations of it

b. A list of 432 built-in passwords that Morris thought to be likely candidates c. All the words in the local system directory

2. It exploited a bug in the finger protocol, which reports the whereabouts of a remote user.

3. It exploited a trapdoor in the debug option of the remote process that receives and sends mail.

If any of these attacks succeeded, the worm achieved communication with the operating system command interpreter.

Recent Worm Attacks

In late 2001, a more versatile worm appeared, known as Nimda. Nimda spreads by multiple mechanisms:

from client to client via e-mail

from client to client via open network shares

from Web server to client via browsing of compromised Web sites

from client to Web server via active scanning for and exploitation of various Microsoft IIS 4.0 / 5.0 directory traversal vulnerabilities

from client to Web server via scanning for the back doors left behind by the "Code Red II" worms

The worm modifies Web documents (e.g., .htm, .html, and .asp files) and certain executable files found on the systems it infects and creates numerous copies of itself under various filenames.

In early 2003, the SQL Slammer worm appeared. This worm exploited a buffer overflow vulnerability in Microsoft SQL server.
Mydoom is a mass-mailing e-mail worm that appeared in 2004

The ideal solution to the threat of viruses is prevention: The next best approach is to be able to do the following:

Detection: Once the infection has occurred, determine that it has occurred and locate the virus.

Identification: Once detection has been achieved, identify the specific virus that has infected a program.

Removal: Once the specific virus has been identified, remove all traces of the virus from the infected program and restore it to its original state. Remove the virus from all infected systems so that the disease cannot spread further.

If detection succeeds but either identification or removal is not possible, then the alternative is to discard the infected program and reload a clean backup version.

There are four generations of antivirus software:

First generation: simple scanners

Second generation: heuristic scanners

Third generation: activity traps

Fourth generation: full-featured protection

A first-generation scanner requires a virus signature to identify a virus.. Such signature- specific scanners are limited to the detection of known viruses. Another type of first-generation scanner maintains a record of the length of programs and looks for changes in length.

A second-generation scanner does not rely on a specific signature. Rather, the scanner uses heuristic rules to search for probable virus infection. One class of such scanners looks for fragments of code that are often associated with viruses.

Another second-generation approach is integrity checking. A checksum can be appended to each program. If a virus infects the program without changing the checksum, then an integrity check will catch the change. To counter a virus that is sophisticated enough to change the checksum

when it infects a program, an encrypted hash function can be used. The encryption key is stored separately from the program so that the virus cannot generate a new hash code and encrypt that. By using a hash function rather than a simpler checksum, the virus is prevented from adjusting the program to produce the same hash code as before.

Third-generation programs are memory-resident programs that identify a virus by its actions rather than its structure in an infected program. Such programs have the advantage that it is not necessary to develop signatures and heuristics for a wide array of viruses. Rather, it is necessary only to identify the small set of actions that indicate an infection is being attempted and then to intervene.

Fourth-generation products are packages consisting of a variety of antivirus techniques used in conjunction. These include scanning and activity trap components. In addition, such a package includes access control capability, which limits the ability of viruses to penetrate a system and then limits the ability of a virus to update files in order to pass on the infection.

The arms race continues. With fourth-generation packages, a more comprehensive defense strategy is employed, broadening the scope of defense to more general-purpose computer security measures.

Advanced Antivirus Techniques

More sophisticated antivirus approaches and products continue to appear. In this subsection, we highlight two of the most important.

Generic Decryption

Generic decryption (GD) technology enables the antivirus program to easily detect even the most complex polymorphic viruses, while maintaining fast scanning speeds . In order to detect such a structure, executable files are run through a GD scanner, which contains the following elements:

CPU emulator: A software-based virtual computer. Instructions in an executable file are interpreted by the emulator rather than executed on the underlying processor. The emulator includes software versions of all registers and other processor hardware, so that the underlying processor is unaffected by programs interpreted on the emulator.

Virus signature scanner: A module that scans the target code looking for known virus signatures.

Emulation control module: Controls the execution of the target code.

Digital Immune System

The digital immune system is a comprehensive approach to virus protection developed by IBM]. The motivation for this development has been the rising threat of Internet-based virus propagation.Two major trends in Internet technology have had an increasing impact on the rate of virus propagation in recent years:

Integrated mail systems: Systems such as Lotus Notes and Microsoft Outlook make it very simple to send anything to anyone and to work with objects that are received.

Mobile-program systems: Capabilities such as Java and ActiveX allow programs to move on their own from one system to another.

1. A monitoring program on each PC uses a variety of heuristics based on system behavior, suspicious changes to programs, or family signature to infer that a virus may be present. The monitoring program forwards a copy of any program thought to be infected to an administrative machine within the organization.

2. The administrative machine encrypts the sample and sends it to a central virus analysis machine.

3. This machine creates an environment in which the infected program can be safely run for analysis. Techniques used for this purpose include emulation, or the creation of a protected environment within which the suspect program can be executed and monitored. The virus analysis machine then produces a prescription for identifying and removing the virus.

4. The resulting prescription is sent back to the administrative machine.

5. The administrative machine forwards the prescription to the infected client.

6. The prescription is also forwarded to other clients in the organization.

7. Subscribers around the world receive regular antivirus updates that protect them from the new virus.

The success of the digital immune system depends on the ability of the virus analysis machine to detect new and innovative virus strains. By constantly analyzing and monitoring the viruses found in the wild, it should be possible to continually update the digital immune software to keep up with the threat.

Behavior-Software

Unlike heuristics or fingerprint-based scanners, behavior-blocking software integrates with the operating system of a host computer and monitors program behavior in real-time for malicious actions. Monitored behaviors can include the following:

Attempts to open, view, delete, and/or modify files;
Attempts to format disk drives and other unrecoverable disk operations;
Modifications to the logic of executable files or macros;
Modification of critical system settings, such as start-up settings;
Scripting of e-mail and instant messaging clients to send executable content; and
Initiation of network communications.

If the behavior blocker detects that a program is initiating would-be malicious behaviors as it runs, it can block these behaviors in real-time and/or terminate the offending software. This gives it a fundamental advantage over such established antivirus detection techniques as fingerprinting or heuristics.