T.Y.B.SC.(IT)

SEMESTER-V (CBCS)

NETWORK SECURITY

SUBJECT CODE : USIT 301

© UNIVERSITY OF MUMBAI

Prof. Suhas Pednekar
Vice Chancellor
University of Mumbai, Mumbai.

Prof. Ravindra D. Kulkarni
Pro Vice-Chancellor,
University of Mumbai.

Prof. Prakash Mahanwar
Director
IDOL, University of Mumbai.

Programe Co-ordinator : **Mandar L. Bhanushe**
Head, Faculty of Science and Technology,
IDOL, University of Mumbai – 400098.

Course Co-ordinator : **Sumedh Pandit Shejole,**
Assistant Professor,
B.Sc.(Information Technology),
Institute of Distance & Open Learning,
University of Mumbai- 400098.

Course Writers : **Sujata Rizal**
Assistant Professor
Kotian S.M.Shetty College Powai

: **Priya Jadhav**
Assistant Professor
N.G.Acharya & D.K.Marathe College

: **Prashant Londhe**
Assistant Professor
Gogate-Jogalekar College, Ratnagiri

: **Geeta Sahu**
Assistant Professor
VSIT

September 2021, Print I

Published by : **Director**
Institute of Distance and Open Learning ,
University of Mumbai,
Vidyanagari, Mumbai - 400 098.

DTP Composed : Varda Offset and Typesetters
Andheri (W), Mumbai - 400 053.

Printed by :

CONTENTS

*****

Syllabus

M. Sc (Information Technology)		Semester – I	
Paper I; SUBJECT: Network Security		Course Code: USIT301	
Periods per week	Lecture	5	
1 Period is 50 minutes	TW/Tutorial/Practical	3	
		Hours	Marks
Evaluation System	Theory Examination	2	60
	TW/Tutorial/Practical	-	40

Unit	Details	Lectures
I	Cryptography: Introduction: Some Simple Cryptosystems, The Shift Cipher, The Substitution Cipher, The Affine Cipher, The Vigenere Cipher, The Hill Cipher, The Permutation Cipher, Stream Ciphers, Cryptanalysis, Cryptanalysis of the Affine Cipher, Cryptanalysis of the Substitution Cipher, Cryptanalysis of the Vigenere Cipher, Cryptanalysis of the LFSR-based Stream Cipher. Shannon's Theory, Perfect Secrecy, Entropy, Huffman Encodings and Entropy, Properties of Entropy, Spurious Keys and Unicity Distance The Data Encryption Standard, Description of DES, An Example of DES Encryption, The DES Controversy, DES in Practice, DES Modes of Operation, A Time-memory Trade-off, Differential Cryptanalysis, An Attack on a 3-round DES, An Attack on a 6-round DES. Introduction to Public-key Cryptography, More Number Theory, The Euclidean Algorithm, The Chinese Remainder Theorem, Other Useful Facts, The RSA Cryptosystem, Implementing RSA, Probabilistic Primality Testing, Attacks On RSA, The Decryption Exponent, Partial Information Concerning Plaintext Bits, The Rabin Cryptosystem, Factoring Algorithms, The p - 1 Method, Dixon's Algorithm and the Quadratic Sieve, Factoring Algorithms in Practice	12
II	**Signature Schemes** : Introduction, The ElGamal Signature Scheme, The Digital Signature Standard, One-time Signatures, Undeniable Signatures, Fail-stop Signatures **Hash Functions** Signatures and Hash Functions, Collision-free Hash Functions The Birthday Attack, A Discrete Log Hash Function, Extending Hash Functions, Hash Functions from Cryptosystems, The MD4 Hash Function, Timestamping. **Key Distribution and Key Agreement** Introduction, Key Predistribution, Blom's Scheme,	12

	Diffie-Hellman Key Predistribution, Kerberos, Diffie-Hellman Key Exchange, The Station-to-station Protocol, MTI Key Agreement Protocols, Key Agreement Using Self-certifying Keys.	
III	Security Trends, The OSI Security Architecture, Security Attacks, Security Services, Security Mechanisms, A Model for Network Security	**12**
IV	Authentication Applications: Kerberos, X.509 Authentication Service, Public-Key Infrastructure, Recommended Reading and Web Sites, Key Terms, Review Questions, and Problems, A Kerberos Encryption Techniques, Electronic Mail Security, Pretty Good Privacy, S/MIME, Key Terms, Review Questions, and Problems, A Data Compression Using Zip, Radix-64 Conversion, PGP Random Number Generation	**12**
V	**IP Security:** IP Security Overview, IP Security Architecture, Authentication Header, Encapsulating Security Payload, Combining Security Associations, Key Management, Recommended Reading and Web Site, Key Terms, **Web Security:** Web Security Considerations, Secure Socket Layer and Transport Layer, Security, Secure Electronic Transaction.	**12**
VI	**Intruders:** Intrusion Detection, Password Management, Recommended Reading and Web Sites. **Malicious Software:** Viruses and Related Threats, Virus Countermeasures, Distributed Denial of Service Attacks. **Firewalls:** Firewall Design Principles, Trusted Systems, Common Criteria for Information Technology Security Evaluation.	

Books and References:
Books: **Cryptography: Theory and Practice,** *Douglas Stinson,* CRC Press, CRC Press LLC (Unit I and II) Cryptography and Network Security Principles and Practices, Fourth Edition, William Stallings, PHI(Pearson), (Unit: III-VI) **References**: Information Security and cyber laws, Saurabh Sharma, student series, Vikas publication. Encryption, Ankit Fadia and J. Bhattacharjee, Vikas publication
Term Work: **Assignments:** Should contain at least 6 assignments (one per unit) covering the Syllabus.

Practical List:

1 Substitution Techniques

a Write a program to perform substitution ciphers to encrypt the plain text to Caesar cipher and to decrypt it back to plain text.

b Write a program to perform substitution ciphers to encrypt the plain text to Modified Caesar cipher and to decrypt it back to plain text.

c Write a program to perform substitution ciphers to encrypt the plain text to homophonic cipher and to decrypt it back to plain text.

d Write a program to perform substitution ciphers to encrypt the plain text to monoalphabetic cipher and to decrypt it back to plain text.

e Write a program to perform substitution ciphers to encrypt the plain text to homophonic cipher and to decrypt it back to plain text.

f Write a program to perform substitution ciphers to encrypt the plain text to polyalphabetic cipher and to decrypt it back to plain text.

2 Transposition Ciphers:

a Write a program to perform transposition ciphers to encrypt the plain text to cipher and to decrypt it back to plain text using rail fence technique.

b Write a program to perform transposition ciphers to encrypt the plain text to cipher and to decrypt it back to plain text using Simple Columnar technique.

c Write a program to perform transposition ciphers to encrypt the plain text to cipher and to decrypt it back to plain text using Columnar with multiple rounds.

D Write a program to encrypt a plain text to a cipher text and decrypt it back to plain text using vernam cipher as the transposition technique

3 Write a program to generate Symmetric Keys for the following Cipher algorithms DES, AES, Blowfish, TripleDES, HmacMD5 and HmacSHA1.

4 Write a program to generate assymmetric Keys for the following Cipher algorithms a) DSA (Digital Signature Algorithm), b) DH (DiffieHellman), c) RSA.

5 Write a program to encrypt input string by using SecretKey of the following algorithms, and then decrypt the encrypted string and compare the decrypted string with the input string. Use the following algorithms for encryption and decryption:

a. DES
b. BlowFish
c. IDEA
d. Triple DES

6 Write a program to encrypt input string by using SecretKey of the following algorithms, and then decrypt the encrypted string and compare the decrypted string with the input string. Use the following algorithms for encryption and decryption:

a. RSA
b. AES
c. DSA

7	Implement following HashFunctions: RSHash, JSHash, BKDRHash, SDBMHash, DJBHash
8	Write a program to encrypt the given string by using RC4 , MD5, algorithms.
9	Write a program for creating, exporting and validating Digital Certificate.
10	Create a permission that controls access to pages of a book. The permission name consists of a book id, a colon, and a set of allowable pages.

*****

UNIT I

1

CRYPTOGRAPHY

Unit structure

1.0 ENCRYPTION

- Encryption provides secure communications in the presence of malicious third parties known as adversaries. Encryption uses an algorithm and a key to transform input (i.e. plain text) into encrypted

output (i.e. ciphertext). A given algorithm will always transform the same plaintext into the same ciphertext if the same key is used.

- The algorithms are considered safe if an attacker cannot determine any properties of the plaintext or key given the ciphertext. An attacker should not be able to determine anything about a key given a large number of plaintext / ciphertext combinations that used the key.

What is the difference between symmetric and asymmetric cryptocurrency?:

- With symmetric encryption, the same key is used for both encryption and decryption. A sender and a recipient must already have a shared key that they both know. Key distribution is a complicated issue and was the impetus for the development of asymmetric cryptography.
- With asymmetric encryption, two different keys are used for encryption and decryption. Each user in an asymmetric cryptosystem has a public key and a private key. The private key is always kept secret, but the public key can be freely distributed.
- Data encrypted with a public key can only be decrypted with the corresponding private key. Hence, sending a message to John requires encrypting the message with John's public key. Only John can decrypt the message, as only John has his private key. Data encrypted with a private key can only be decrypted with the corresponding public key. Likewise, Jane could digitally sign a message with her private key, and anyone with Jane's public key could decrypt the signed message and verify that Jane sent it.
- Symmetric is generally very fast and ideal for encrypting large amounts of data (for example, an entire disk partition or database). Asymmetric is much slower and can only encrypt data smaller than the key size (typically 2048 bits or less). Therefore, asymmetric encryption is generally used to encrypt symmetric encryption keys which are then used to encrypt much larger chunks of data. For digital signatures, asymmetric encryption is typically used to encrypt message hashes rather than entire messages.
- A cryptosystem provides cryptographic key management, including key generation, exchange, storage, use, revocation, and replacement.

What problems does cryptocurrencies solve?:

- A secure system must provide various guarantees, such as confidentiality, integrity and availability of data, as well as authenticity and non-repudiation. When used correctly, encryption helps provide these guarantees. Encryption can ensure the confidentiality and integrity of both data in transit and data at rest. You can also authenticate senders and recipients against each other and protect yourself from repudiation.
- Software systems typically have multiple endpoints, typically multiple clients and one or more back-end servers. These client / server

communications occur over networks that cannot be trusted. Communication takes place over open public networks such as the Internet or private networks that can be compromised by external attackers or malicious users.

- Communications that cross untrusted networks can be protected. There are two main types of attacks that an opponent can attempt to perform on a network. Passive attacks involve an attacker who simply intercepts a network segment and attempts to read sensitive information while traveling. Passive attacks can be online (where an attacker reads traffic in real time) or offline (where an attacker simply captures traffic in real time and sees it later, perhaps after spending some time at decrypt it). Active attacks involve an attacker posing as a client or server,
- The confidentiality and integrity protocols provided by cryptographic protocols such as SSL / TLS can protect communications from malicious eavesdropping and tampering. Authenticity protections ensure that users are actually communicating with systems as intended. For example, are you sending your online banking password to your bank or someone else?
- It can also be used to protect data at rest. Data on a removable disk or in a database can be encrypted to prevent the disclosure of confidential data if physical media is lost or stolen. Furthermore, it can also provide integrity protection of inactive data to detect malicious tampering.

What are the principles?:

- The most important principle to keep in mind is that you should never attempt to design your own cryptosystem. The brightest cryptographers in the world (including Phil Zimmerman and Ron Rivest) systematically create ciphers with serioussecurity flawsin them. For a cryptosystem to be considered "secure", it must face careful scrutiny bysecurity community. Never trust security through obscurity or the fact that attackers may not be aware of your system. Remember that bad guys and determined attackers will try to attack your system.
- The only things that should be "secret" when it comes to a secure cryptographic system are the keys themselves. Make sure you take the appropriate steps to protect the keys used by your systems. Never store clear-text encryption keys along with the data they protect. It's like closing the front door and putting the key under the doormat. It is the first place an attacker will look at. Here are three common ways to protect keys (from least secure to most secure):
    1. Store keys in a file system and protect them with robust access control lists (ACLs). Remember to adhere to the principle of least privilege.
    2. Encrypt your data encryption keys (DEK) with a second key encryption key (KEK). The KEK must be generated using

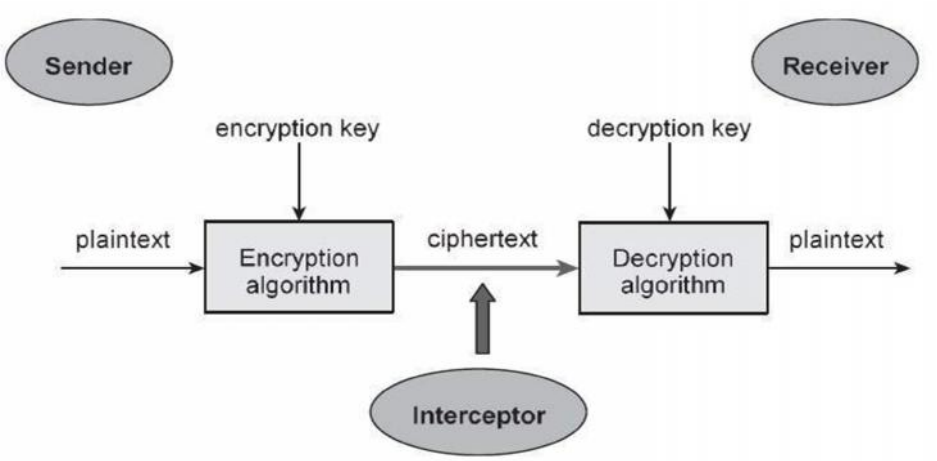
Password Based Encryption (PBE). A password known to a minimum number of administrators can be used to generate a key using an algorithm such as bcrypt, scrypt, or PBKDF2 and used to start the cryptographic system. This eliminates the need to store the erase key anywhere.

3. A Hardware Security Module (HSM) is a tamper-proof hardware device that can be used to securely store keys. The code can make API calls to an HSM to provide keys when needed or to perform data decryption in the HSM itself.

1.1 CRYPTOSYSTEMS

A cryptosystem is an implementation of cryptographic techniques and the infrastructure that accompanies them to provide information security services. A cryptographic system is also known as an encryption system.

Let's analyze a simple cryptosystem model that provides confidentiality to the information transmitted. This basic model is shown in the following illustration:



The illustration shows a sender who wants to transfer some confidential data to a recipient in such a way that any interlocutor or interception on the communication channel cannot extract the data.
The goal of this simple cryptographic system is that at the end of the process, only the sender and recipient know the plaintext.

Components of a cryptosystem:

The various components of a basic cryptosystem are as follows:

- Plain text. These are the data to be protected during transmission.
- Encryption algorithm. It is a mathematical process that produces ciphertext for any plaintext and encryption key. It is a cryptographic

algorithm that accepts plain text and an encryption key as input and produces encrypted text.

- Encrypted text. It is the encrypted version of the plaintext produced by the encryption algorithm that uses a specific encryption key. The ciphertext is not protected. Scrolls in public channels. It can be intercepted or compromised by anyone with access to the communication channel.
- The decryption algorithm is a mathematical process that produces a unique plaintext for any ciphertext and decryption key. It is a cryptographic algorithm that accepts a ciphertext and a decryption key as input and generates plain text. The decryption algorithm basically reverses the encryption algorithm and is therefore closely related to it.
- Encryption key. It is a value known to the sender. The sender enters the encryption key into the encryption algorithm along with the plaintext to compute the ciphertext.
- Decryption key. It is a value known to the recipient. The decryption key is related to the encryption key, but it is not always identical to it. The recipient enters the decryption key into the decryption algorithm along with the ciphertext to calculate the plaintext.

For a given cryptographic system, a collection of all possible decryption keys is called the key space.

An interceptor (an attacker) is an unauthorized entity that attempts to determine plain text. You can view the ciphertext and learn about the decryption algorithm. However, you should never know the decryption key.

Types of cryptosystems:

Basically, there are two types of cryptosystems based on the way encryption-decryption is performed on the system:

- Symmetric key encryption
- Asymmetric key encryption

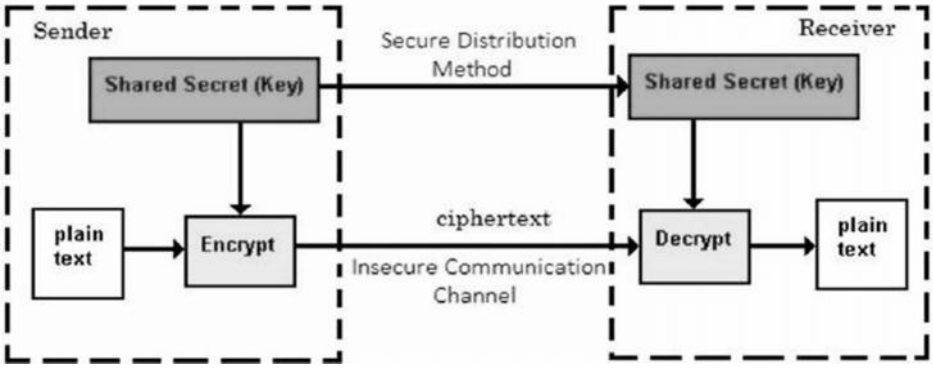
The main difference between these cryptographic systems is the relationship between the encryption and the decryption key. Logically, in any cryptographic system, both keys are closely associated. It is practically impossible to decrypt the ciphertext with the key that is not related to the encryption key.

Symmetric key encryption:

The encryption process in which the same keys are used to encrypt and decrypt information is known as symmetric key encryption.

The study of symmetric cryptosystems is known as symmetric cryptography. Symmetric cryptosystems are sometimes also called secret key cryptosystems.

Some known examples of symmetric key encryption methods are: Digital Encryption Standard (DES), Triple-DES (3DES), IDEA, and BLOWFISH.



Before 1970, all cryptographic systems used symmetric key cryptography. Even today, its relevance is very high and it is widely used in many cryptographic systems. This encryption is highly unlikely to disappear as it has some advantages over asymmetric key encryption.

The most important features of the cryptosystem based on symmetric key cryptography are:

- People who use symmetric key cryptography must share a common key before exchanging information.
- It is recommended to change the keys regularly to avoid any attack on the system.
- A robust mechanism is needed for key exchange between communicating parties. Since the keys have to be changed regularly, this mechanism becomes expensive and cumbersome.
- In a group of n people, to allow communication between any two people, the number of keys required for the group is $n \times (n - 1) / 2$.
- The key length (number of bits) in this encryption is shorter and therefore the encryption-decryption process is faster than asymmetric key encryption.
- The processing power of a computer system required to run a symmetric algorithm is less.

Challenge of the symmetric key cryptography system:

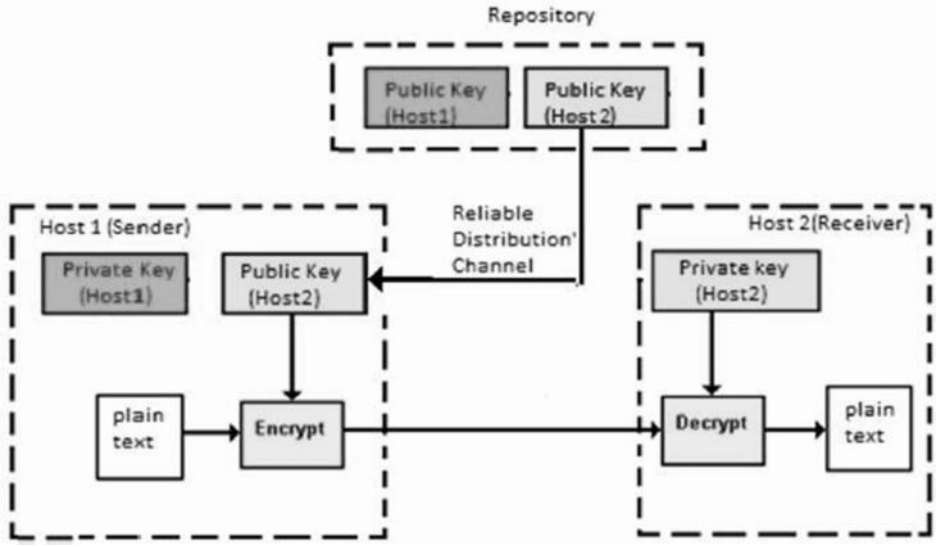
There are two restrictive challenges in using symmetric key cryptography.

- Establishment of the key: Before any communication, both the sender and the recipient must agree on a secret symmetric key. Requires a secure key creation mechanism.
- Trust issue: Since the sender and recipient use the same symmetric key, there is an implicit requirement that the sender and recipient "trust" each other. For example, it may happen that the recipient has lost the key of an attacker and the sender is not informed.

These two challenges are very restrictive for modern communication. Today, people need to exchange information with unknown and unreliable parties. For example, the communication between online sellers and customers. These limitations of symmetric key cryptography have led to asymmetric key cryptographic schemes.

Asymmetric key encryption:

The encryption process in which different keys are used to encrypt and decrypt information is known as asymmetric key cryptography. Although the keys are different, they are mathematically related and therefore it is possible to recover the plaintext by decrypting the ciphertext. The process is shown in the following figure:



Asymmetric key cryptography was invented in the 20th century to overcome the need for a secret pre-shared key between communicating people. The most important features of this encryption scheme are as follows:

- All users of this system must have a different key pair, private key, and public key. These keys are mathematically related: when one key is used for encryption, the other can decrypt the ciphertext back to its original plaintext.

- It requires you to put the public key in a public repository and the private key as a well-kept secret. Therefore, this encryption scheme is also called public key cryptography.
- Although the user's public and private keys are related, it is not computationally possible to find each other. This is one of the strengths of this scheme.
- When Host1 needs to send data to Host2, it obtains Host2's public key from the repository, encrypts the data, and transmits.
- Host2 uses your private key to extract plain text.
- The length of the keys (number of bits) in this encryption is large and therefore the encryption-decryption process is slower than symmetric key encryption.
- The processing power of a computer system required to run an asymmetric algorithm is greater.

Symmetric cryptosystems are a natural concept. Conversely, public-key cryptosystems are rather difficult to understand.

You may be thinking, how can the encryption key and the decryption key be "related" and yet is it impossible to determine the decryption key from the encryption key? The answer is in the mathematical concepts. It is possible to design a cryptosystem whose keys have this property. The concept of public key cryptography is relatively new. Fewer public key algorithms are known than symmetric algorithms.

Challenge of the public key cryptographic system:

Public-key cryptographic systems have a significant challenge: the user must trust that the public key they are using in communications with a person is actually that person's public key and has not been forged by malicious third parties.

This is typically accomplished through a public key infrastructure (PKI) made up of a trusted third party. The third party securely manages and guarantees the authenticity of the public keys. When the third party is asked to provide the public key for any communicating person X, they rely on them to provide the correct public key.

The third party is satisfied with the user's identity through the attestation process, notarization, or some other process: whether X is the only, or globally unique, X. The most common method of making verified public keys available is to embed them in a certificate digitally signed by a trusted third party.

Relationship between encryption schemes:

Here is a summary of the basic key properties of two types of cryptosystems:

	Symmetric cryptosystems	**Public key cryptosystems**
Relationship between the keys	Same	Different, but mathematically related
Encryption key	Symmetrical	Public
Decryption key	Symmetrical	Private

Due to the advantages and disadvantages of both systems, symmetric key and public key cryptographic systems are often used together in practical information security systems.

Kerckhoff's principle for the cryptosystem:

In the 19th century, a Dutch cryptographer A. Kerckhoff provided the requirements for a good cryptographic system. Kerckhoff said that a cryptographic system should be secure even if everything about the system, except the key, is in the public domain. The six design principles defined by Kerckhoff for the cryptosystem are:

- The cryptosystem should be practically indestructible, if not mathematically.
- The fall of the cryptosystem into the hands of an intruder must not involve any compromise of the system, avoiding inconvenience for the user.
- The key must be easily communicable, memorable and modifiable.
- The ciphertext must be transmissible by telegraph, an insecure channel.
- The encryption device and documents must be portable and managed by one person.
- Finally, the system must be easy to use, requiring no mental effort or knowledge of a long set of rules to follow.

The second rule is known today as the Kerckhoff principle. It is applied practically in all contemporary encryption algorithms such as DES, AES, etc. These public algorithms are considered completely safe. The security of the encrypted message depends solely on the security of the secret encryption key.

Keeping algorithms secret can be a significant barrier to cryptanalysis. However, keeping algorithms secret is only possible if they are used in a strictly limited circle.

In the modern era, cryptocurrencies must satisfy internet-connected users. In such cases, the use of a secret algorithm is not feasible, so Kerckhoff's principles have become essential guidelines for algorithm design in modern cryptography.

1.2 CAESAR'S CIPHER TECHNIQUE

Caesar encryption is the simplest and oldest encryption method. The Caesar encryption method is based on single-alphabet encryption and is also called turn-based encryption or additive encryption. Julius Caesar used turn-based cryptography (additive cryptography) to communicate with his officers. For this reason, the turn-based cipher technique is called Caesar cipher. The Caesar cipher is a kind of substitution (substitution) cipher, in which all letters in the plaintext are replaced by another letter.
Let's take an example to understand Caesar's cipher, suppose we pass with 1, then A will be replaced by B, B will be replaced by C, C will be replaced by D, D will be replaced by C and this process continues. until all plain text is finished.

Caesar ciphers are a weak encryption method. It can be easily hacked. It means that the message encrypted with this method can be easily decrypted.

Plain text: It is a simple message written by the user.

Encrypted text: It is an encrypted message after applying some technique.

The encryption formula is:

In (x) = (x + n) mod 26

The decryption formula is:

Dn (x) = (xi - n) mod 26

However, if the value (Dn) becomes negative (-ve), in this case we will add 26 to the negative value. Where is it,

E indicates encryption

D indicates decryption

x indicates the value of the letters

n denotes the key value (exchange value)

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

Example 1 Use Caesar encryption to encrypt and decrypt the "JAVATPOINT" message and the key value (offset) for this message is 3.

Encryption:

We apply encryption formulas by character, in alphabetical order. The encryption formula is: En (x) = (x + n) mod 26

Plain text: J → 09	It is: (09 + 3) mod 26	Ciphertext: 12 → M
Plain text: A → 00	It is: (00 + 3) mod 26	Ciphertext: 3 → D
Plain text: V → 21	It is: (21 + 3) mod 26	Ciphertext: 24 → Y
Plain text: A → 00	It is: (00 + 3) mod 26	Ciphertext: 3 → D
Plain text: V → 19	It is: (19 + 3) mod 26	Ciphertext: 22 → W.
Plain text: P → 15	It is: (15 + 3) mod 26	Ciphertext: 18 → S
Plain text: O → 14	It is: (14 + 3) mod 26	Ciphertext: 17 → R
Plain text: I → 08	It is: (08 + 3) mod 26	Ciphertext: 11 → L
Plain text: N → 13	It is: (13 + 3) mod 26	Ciphertext: 16 → Q
Plain text: V → 19	It is: (19 + 3) mod 26	Ciphertext: 22 → W.

The encrypted message is "MDYDWSRLQW". Note that Caesar's cipher is mono-alphabetic, so the same plain text letters are encrypted as the same letters. For example, "JAVATPOINT" has "A", encrypted with "D".

Decoded:

We apply decryption formulas by character, in alphabetical order.
The decryption formula is: Dn (x) = (xi - n) mod 26

However, if the value (Dn) becomes negative (-ve), in this case we will add 26 to the negative value.

Ciphertext: M → 12	Dn: (12-3) mod 26	Plain text: 09 → J
Ciphertext: D → 03	Dn: (03 - 3) mod 26	Plain text: 0 → A
Ciphertext: Y → 24	Dn: (24-3) mod 26	Plain text: 21 → V
Plain text: A → 00	It is: (00 + 3) mod 26	Ciphertext: 3 → D
Plain text: V → 19	It is: (19 + 3) mod 26	Ciphertext: 22 → W.
Plain text: P → 15	It is: (15 + 3) mod 26	Ciphertext: 18 → S
Plain text: O → 14	It is: (14 + 3) mod 26	Ciphertext: 17 → R
Plain text: I → 08	It is: (08 + 3) mod 26	Ciphertext: 11 → L
Plain text: N → 13	It is: (13 + 3) mod 26	Ciphertext: 16 → Q
Plain text: V → 19	It is: (19 + 3) mod 26	Ciphertext: 22 → W.

The decrypted message is "JAVATPOINT".

Example: 2 It uses Caesar's cipher to encrypt and decrypt the "HELLO" message and the key (modification) value for this message is 15.

Encryption:

We apply encryption formulas by character, in alphabetical order.
The encryption formula is: En (x) = (x + n) mod 26

Plain text: H → 07	It is: (07 + 15) mod 26 mod	Ciphertext: 22 → W.
Plain text: E → 04	It is: (04 + 15) mod 26 mod	Ciphertext: 19 → T
Plain text: L → 11	It is: (11 + 15) mod 26	Ciphertext: 00 → LA
Plain text: L → 11	It is: (11 + 15) mod 26	Ciphertext: 00 → LA
Plain text: O → 14	It is: (14 + 15) mod 26	Ciphertext: 03 → D

Note that Caesar's cipher is mono-alphabetic, so the same plain text letters are encrypted as the same letters. For example, "HELLO" has "L", encrypted by "A".

The encrypted message for this plaintext is "WTAAD".

Decoded:

We apply decryption formulas by character, in alphabetical order.
The decryption formula is: Dn (x) = (xi - n) mod 26

Ciphertext: W → 22	Dn: (22-15) mod 26	Plain text: 07 → H.
Ciphertext: T → 19	DN: (19-15) mod 26	Plain text: 04 → E
Ciphertext: A → 00	Dn: (00-15) mod 26	Plain text: 11 → L
Ciphertext: A → 00	Dn: (00-15) mod 26	Plain text: 11 → L
Ciphertext: D → 03	Dn: (03-15) mod 26	Normal text: 14 → O

The decrypted message is "HELLO".

Note: If in any case the value (Dn) becomes negative (-ve), in this case we will add 26 to the negative value. Like, the third letter of the ciphertext.

Dn = (00-15) mod 26

= -15

The value of dn is negative, so 26 will be added to it.

= -15 + 26

= 11

Advantages of Caesar encryption:

Its advantages are as follows:
How to find yet another highest salary in SQL

1. It is very easy to implement.
2. This method is the simplest encryption method.

3. Only a short key is used in the whole process.
4. If a system does not use complex coding techniques, this is the best way to do it.
5. It requires only a few computing resources.

Disadvantages of Caesar encryption:

Its disadvantages are as follows: -

1. It can be easily hacked. It means that the message encrypted with this method can be easily decrypted.
2. Provides very little security.
3. By looking at the pattern of the letters on it, the entire message can be decrypted.

1.3 SIMPLE REPLACEMENT ENCRYPTION

The simple substitution cipher is the most widely used cipher and includes an algorithm to substitute each plaintext character for each ciphertext character. In this process, the alphabets are confused with respect to Caesar's encryption algorithm.

Example:

Keys for simple surrogate encryption typically consist of 26 letters. An example of a key is the simple alphabet: abcdefghijklmnopqrstuvwxyz encrypted alphabet: phqgiumeaylnofdxjkrcvstzwb

An example of encryption using the above key is:

plain text: defend the east wall of the castle

ciphertext: giifg cei iprc tpnn du cei qprcni

The following code shows a program to implement simple substitution cryptography:

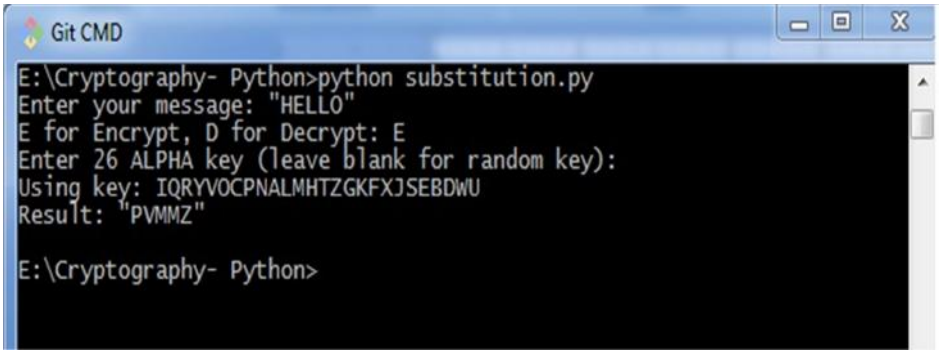
```
import random, sys

LETTERS = 'ABCDEFGHHIJKLMNOPQRSTUVWXYZ'

main def ():

 message = ''

 if len (sys.argv)> 1:

 with open (sys.argv [1], 'r') as f:

 message = read ()

 the rest:
```

```
message = raw_input ("Enter your message:")
mode = raw_input ("E to encrypt, D to decrypt:")
key = "

while checkKey (key) is False:
key = raw_input ("Enter 26 ALPHA keys (leave blank for random key):")
if key == ":
key = getRandomKey ()
if checkKey (key) is False:
                print ("There is an error in the key or symbol set.")
translate = translateMessage (message, key, mode)
print ('Using key:% s'% (key))
if len (sys.argv)> 1:
fileOut = 'enc.' + sys.argv [1]
with open (fileOut, 'w') as f:
f.write (translated)
print ('Success! File written to:% s'% (fileOut))
else: print ('Result:' + translated)
# Save the key in the list, sort it, convert it again, compare it with the alphabet.
def checkKey (key):
keyString = " .join (sorted (list (key)))
return keyString == LETTERS
def translateMessage (message, key, mode):
translated = "
charsA = LETTERS
charactersB = key
# If the decryption mode is detected, swap A and B
if mode == 'D':
charsA, charsB = charsB, charsA
for the symbol in the message:
if symbol.upper () in charsA:
symIndex = charsA.find (symbol.upper ())
```

```
 if symbol.isupper ():
 translated + = charsB [symIndex] .upper ()
 the rest:
 translated + = charsB [symIndex] .lower ()
                    the rest:
 translated + = symbol
 back translated
def getRandomKey ():
 randomList = list (LETTERS)
 random.shuffle (random list)
 return " .join (random list)
if __name__ == '__main__':
 main()
```

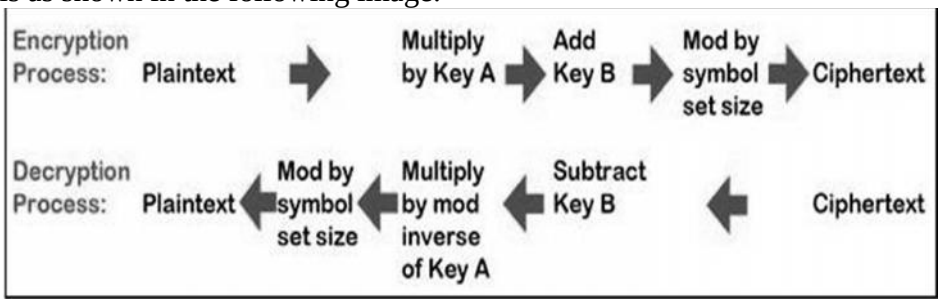
Production:

You can see the following output when you implement the above code:



1.4 AFFINE CRYPTOGRAPHY

Affine Cipher is the combination of the Multiplicative Cipher and Caesar Cipher algorithm. The basic implementation of affine cryptography is as shown in the following image:



In this chapter we will implement affine cryptography by creating its corresponding class which includes two basic functions for encryption and decryption.

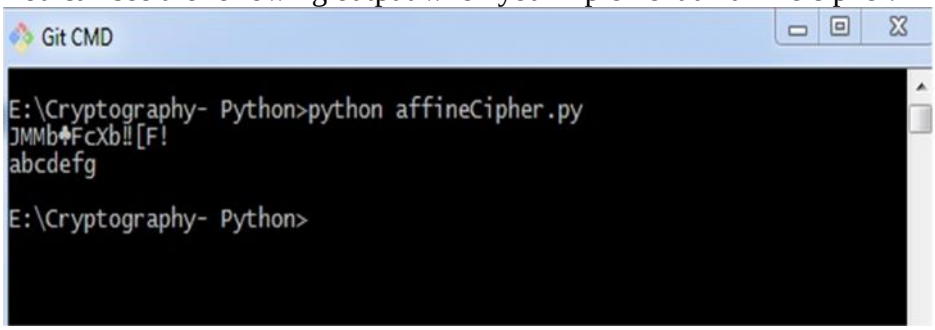
Code:

You can use the following code to implement affine encryption:

```
Affine class (object):
 DEATH = 128
 KEY = (7, 3, 55)
 def __init __ (me):
 To approve
 def encryptChar (self, char):
 K1, K2, kI = self.KEY
 return chr ((K1 * ord (char) + K2)% self.DIE)
 def encrypt (self, string):
 return "" .join (map (self.encryptChar, string))
 def decryptChar (self, char):
 K1, K2, KI = self.KEY
 return chr (KI * (ord (char) - K2)% self.DIE)
 def decrypt (self, string):
 return "" .join (map (self.decryptChar, string))
                related = Affine ()
print affine.encrypt ("Affine encryption")
affine.decrypt print ('* 18? FMT')
```

Production:

You can see the following output when you implement an affine cipher:



The output shows the encrypted message for the Affine Cipher plain text message and the decrypted message for the message sent as input abcdefg.

1.5 VIGENERE ENCRYPTION

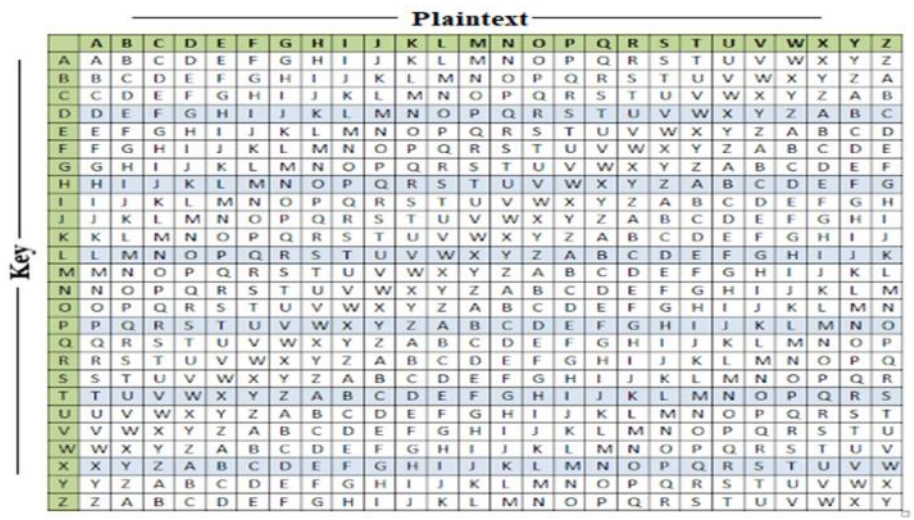
Strong encryption is an algorithm used to encrypt and decrypt text. Vigorous cipher is an alphabetic text encryption algorithm that uses a series of interconnected Caesar ciphers. It is based on the letters of a

keyword. It is an example of a polyalphabetic substitution cipher. This algorithm is easy to understand and implement. This algorithm was first described in 1553 by Giovan Battista Bellaso. Use a Vigenere table or a Vigenere square for text encryption and decryption. The table in effect is also called a straight table.

Two methods perform encryption in effect.

Method 1:

When a valid table is provided, encryption and decryption are performed using the valid table (array 26 * 26) in this method.



Example: the plain text is "JAVATPOINT" and the key is "BEST".

To generate a new key, the given key is repeated in a circular fashion, as long as the length of the plaintext is not equal to the new key.

J	A	V	A	T	P	O	I	N	T
B	E	S	T	B	E	S	T	B	E

Encryption:

The first letter of the plaintext is combined with the first letter of the key. The plaintext column "J" and key row "B" intersect the alphabet for "K" in the current table, so the first letter of the ciphertext is "K".

HTML tutorial:

Similarly, the second letter of the plaintext is combined with the second letter of the key. Plain text column "A" and key row "E" intersect the alphabet for "E" in the current table, so the second letter of the ciphertext is "E".

This process continues continuously until the plaintext is finished.

Encrypted text = KENTUTGBOX

decoded

Decryption is done using the row of keys in the current table. First, select the key letter row, find the position of the ciphertext letter in that row, and then select the corresponding ciphertext column label as plain text.

K	E	N	T	U	T	G	B	O	X
B	E	S	T	B	E	S	T	B	E

For example, in the key line there is "B" and the ciphertext is "K" and this letter of the ciphertext appears in the column "J", which means that the first letter of the plaintext is "J".

The next in the key line is "E" and the ciphertext is "E" and this ciphertext letter appears in the "A" column, which means the second plaintext letter is "A".

This process continues continuously until the end of the ciphertext.

Plain text = JAVATPOINT

Method 2:

When the actual table is not provided, encryption and decryption are performed by Vigenar's algebraic formula in this method (converting letters (AZ) to numbers (0-25)).

The cryptographic formula is,

Ei = (Pi + Ki) mod 26

The decryption formula is,

Di = (Ei - Ki) mod 26

If in any case the value (Di) becomes negative (-ve), in this case, we will add 26 to the negative value.

Where is it,

E indicates encryption.

D indicates decryption.

P indicates normal text.

K indicates the key.

Note: "i" indicates the offset of the i-th number of letters, as shown in the following table.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

Example: the plain text is "JAVATPOINT" and the key is "BEST".
Encryption: Ei = (Pi + Ki) mod 26

Plain text	J	FOR	V.	FOR	T.	P.	OR	I	North	T.
Plain text value (P)	09	00	twenty-one	00	19	fifteen	14	08	13	19
Key	B.	myself	S.	T.	B.	myself	S.	T.	B.	myself
Key value (K)	01	04	18	19	01	04	18	19	01	04
Ciphertext value (E)	10	04	13	19	twenty	19	06	01	14	2. 3
Encrypted text	K.	myself	North	T.	OR	T.	GRAM	B.	OR	X

decoded: Di = (Ei - Ki) mod 26

If in any case the value (Di) becomes negative (-ve), in this case, we will add 26 to the negative value. Come, the third letter of the ciphertext;

N = 13 and S = 18

Di = (Ei - Ki) mod 26

Di = (13 - 18) mod 26

Di = -5 mod 26

Di = (-5 + 26) mod 26

Di = 21

Encrypted text	K.	myself	North	T.	OR	T.	GRAM	B.	OR	X
Ciphertext value (E)	10	04	13	19	twenty	19	06	01	14	2. 3
Key	B	mys	S.	T.	B.	mys	S.	T	B.	mys

	.	elf				elf		.		elf
Key value (K)	01	04	18	19	01	04	18	19	01	04
Plain text value (P)	09	00	twenty-one	00	19	fifteen	14	08	13	19
Plain text	J	FOR	V.	FOR	T.	P.	OR	I	North	T.

1.6 HILL ENCRYPTION

The Hill cipher is a polygraph replacement cipher based on linear algebra. Each letter is represented by a modulo number 26. The simple pattern A = 0, B = 1,…, Z = 25 is often used, but this is not an essential feature of the cipher. To encrypt a message, each block of n letters (considered as a vector of n components) is multiplied by an invertible n × n matrix, against modulo 26. To decrypt the message, each block is multiplied by the inverse of the matrix used for cryptography.

The matrix used for the encryption is the encryption key, and must be chosen randomly from the set of n × n invertible matrices (modulo 26).

Examples:

Input: Plain text: ACT

Key: GYBNQKURP

Output: Ciphertext: POH

Input: Plain text: GFG

Legend: HILLMAGIC

Output: Ciphertext: SWK

Encryption:

We need to encrypt the 'ACT' message (n = 3). The key is "GYBNQKURP" which can be written as an nxn array:

$$\begin{bmatrix} 6 & 24 & 1 \\ 13 & 16 & 10 \\ 20 & 17 & 15 \end{bmatrix}$$

The 'ACT' message is written as a vector:

$$\begin{bmatrix} 0 \\ 2 \\ 19 \end{bmatrix}$$

The encrypted vector is given as:

$$\begin{bmatrix} 6 & 24 & 1 \\ 13 & 16 & 10 \\ 20 & 17 & 15 \end{bmatrix} \begin{bmatrix} 0 \\ 2 \\ 19 \end{bmatrix} = \begin{bmatrix} 67 \\ 222 \\ 319 \end{bmatrix} \equiv \begin{bmatrix} 15 \\ 14 \\ 7 \end{bmatrix} \pmod{26}$$

which matches the ciphertext of 'POH'

decoded

To decrypt the message, we convert the ciphertext back to a vector, then simply multiply by the inverse matrix of the key matrix (IFKVIVVMI in letters). The inverse of the matrix used in the previous example is:

$$\begin{bmatrix} 6 & 24 & 1 \\ 13 & 16 & 10 \\ 20 & 17 & 15 \end{bmatrix}^{-1} \equiv \begin{bmatrix} 8 & 5 & 10 \\ 21 & 8 & 21 \\ 21 & 12 & 8 \end{bmatrix} \pmod{26}$$

For the ciphertext above 'POH':

$$\begin{bmatrix} 8 & 5 & 10 \\ 21 & 8 & 21 \\ 21 & 12 & 8 \end{bmatrix} \begin{bmatrix} 15 \\ 14 \\ 7 \end{bmatrix} \equiv \begin{bmatrix} 260 \\ 574 \\ 539 \end{bmatrix} \equiv \begin{bmatrix} 0 \\ 2 \\ 19 \end{bmatrix} \pmod{26}$$

which returns 'ACT'.

1.7 ENCRYPTION TRANSPOSITION

Transposition Cipher is a cryptographic algorithm in which the order of plain text alphabets is rearranged to form a ciphertext. Actual plain text alphabets are not included in this process.

Example:

A simple example of a transposition cipher is the columnar transposition cipher, in which each character in plain text is written horizontally with a specified alphabetic width. The encryption is written vertically, which creates a completely different ciphertext.

Consider the simple hello world text and apply the simple columnar transposition technique as shown below

h	e	l	l
o	w	o	r
l	d		

Plain text characters are placed horizontally and the ciphertext is created in vertical format as: holewdlo lr. Now the recipient must use the same table to decrypt the ciphertext into plain text.

Code:

The following program code demonstrates the basic implementation of the column transposition technique:

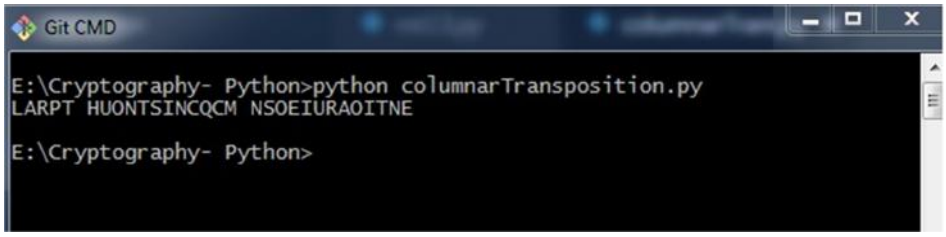
```
def split_len (seq, length):
 return [seq [i: i + length] for i in the interval (0, len (seq), length)]
def encoding (key, plain text):
 order = {
 int (val): num for num, val in enumerate (key)
 }
ciphertext = ''

for the sorted index (order.keys ()):
 per part in split_len (plain text, len (key)):
 proof: ciphertext + = part [order [index]]
 except IndexError:
 Follow
 return the ciphertext
print (encoding ('3214', 'HELLO'))
```

Explanation:

- Using the split_len () function, we can split plain text characters, which can be placed in column or row format.
- The encryption method helps to create the ciphertext with a key that specifies the number of columns and prints the ciphertext by reading the characters in each column.

Production:

The program code for the basic implementation of the column transposition technique provides the following output:



Note: Cryptoanalysts have observed a significant improvement in cryptographic security when the transposition technique is performed. They also noted that re-encoding the ciphertext using the same transposition cipher creates greater security.

1.8 STREAM ENCRYPTION

Difference between block encryption and stream encryption:

Both block and stream encryption are the encryption methods used primarily to directly convert plain text to encrypted text and belong to the symmetric key cryptography family.

Below are the important differences between Block Cipher and Stream Cipher.

No sir.	Key	Block encryption	Stream encryption
1	Definition	Block Cipher is the type of encryption where the plaintext conversion is done by taking your block at a time.	On the other hand, Stream Cipher is the type of encryption where the plaintext conversion is done by taking one byte of the plaintext at a time.
Two	Bit conversion	Since Block Cipher accepts blocks at a time, relatively more bits are converted than Stream Cipher, specifically 64 bits or more can be converted at a time.	On the other hand, in the case of Stream Cipher, a maximum of 8 bits can be converted at a time.
3	Start	Block Cipher uses the principle of confusion and	On the other hand, Stream Cipher only uses the

		diffusion for the conversion required for encryption.	confusion principle for conversion.
4	Algorithm	For clear text encryption, Block Cipher uses Electronic Code Book (ECB) and Cipher Block Chaining (CBC) algorithm.	Stream Cipher uses the CFB (Cipher Feedback) and OFB (Output Feedback) algorithm instead.
5	decoded	A combination of multiple bits is encrypted in the case of Block Cipher, so reverse encryption or decryption is relatively complex compared to Stream Cipher.	On the other hand, Stream Cipher uses XOR for encryption which can be easily restored in plain text.
6	Implementation	The main implementation of Block Cipher is Feistel Cipher.	On the other hand, the main implementation of Stream Cipher is Vernam Cipher.

1.9 ENCRYPTION

Humans of all ages had two intrinsic needs: (a) to communicate and share information and (b) to communicate selectively. These two needs gave rise to the art of encoding messages in such a way that only designated persons could access the information. Unauthorized persons cannot extract any information, even if the encrypted messages fall into their hands.

The art and science of hiding messages to introduce secrecy into information security is recognized as cryptography.

The word "cryptography" was coined by combining two Greek words, "Krypto" which means hidden and "graphene" which means writing.

History of cryptography:

The art of cryptography is considered born along with the art of writing. With the evolution of civilizations, human beings have organized

themselves into tribes, groups and kingdoms. This has led to the emergence of ideas such as power, battles, supremacy and politics. These ideas have further fueled people's natural need to secretly communicate with selective recipients, which in turn has also ensured the continued evolution of cryptocurrencies.

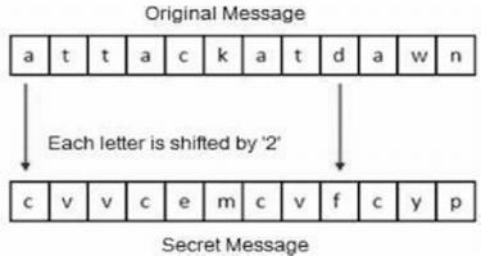
The roots of cryptography lie in the Roman and Egyptian civilizations.

Hieroglyph - the oldest cryptographic technique:

The earliest known evidence of cryptography dates back to the use of "hieroglyphs". About 4,000 years ago, the Egyptians communicated using messages written in hieroglyphics. This code was the secret known only to scribes who used to transmit messages on behalf of kings. Below is one of these hieroglyphs.

Later, scholars switched to the use of simple mono-alphabetic substitution ciphers in the period between 500 and 600 BC. This involved replacing the message alphabets with other alphabets with some secret rule. This rule has become a key to recovering the message from the unreadable message.

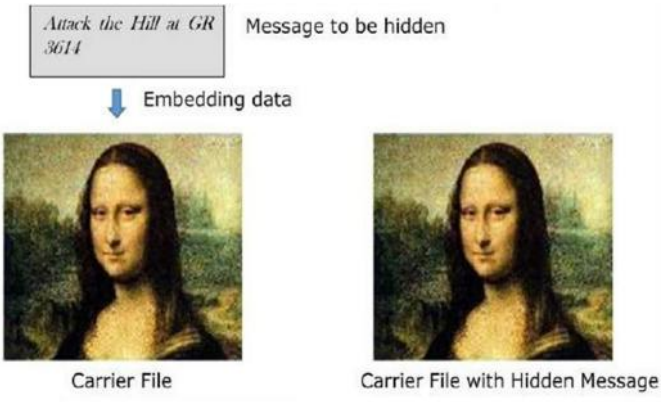
The previous Roman encryption method, popularly known as the Caesar exchange cipher, is based on the exchange of the letters of a message with an agreed number (three was a common choice), the recipient of this message exchanges the letters with the same number. and get the original message.



Steganography:

Steganography is similar but adds another dimension to cryptography. In this method, people not only want to protect the secrecy of information by hiding it, but they also want to make sure that unauthorized people do not get evidence of the information. For example, invisible watermarks.

In steganography, an unintended recipient or intruder is unaware that the observed data contains hidden information. In encryption, an intruder is usually aware that the data is being communicated, because he can see the encrypted / encrypted message.



Evolution of cryptography:

During and after the European Renaissance, various Italian and papal states led the rapid proliferation of cryptographic techniques. At this time, various analysis and attack techniques have been studied to crack the secret codes.

- Improved coding techniques, such as Vigenere Coding, originated in the 15th century, with letters moving in the message with several different points rather than moving them the same number of places.
- Only after the 19th century did cryptography evolve from ad hoc approaches to cryptography to the more sophisticated art and science of information security.
- In the early 20th century, the invention of mechanical and electromechanical machines, such as the Enigma rotor machine, provided more advanced and efficient means of encoding information.
- During the WWII period, both cryptography and cryptanalysis became overly mathematical.

With advances in this field, government organizations, military units and some corporate houses have begun to adopt the applications of cryptography. They used cryptography to protect their secrets from others. Now, the advent of computers and the Internet has put effective encryption within the reach of ordinary people.

Modern cryptography:

Modern cryptography is the cornerstone of cyber and communications security. Its basis is based on various concepts of mathematics, such as number theory, computational complexity theory and probability theory.

Features of modern cryptography:

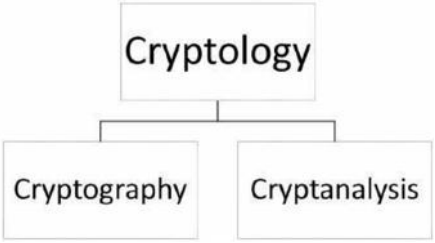
There are three main characteristics that separate modern cryptocurrencies from the classic approach.

classic cryptography	Modern cryptography
Directly manipulate traditional characters - letters and digits.	It works on binary bit streams.
It is mainly based on "safety through the dark". The techniques used for encryption were kept secret and only the parties involved in the communication were aware of them.	It relies on publicly known mathematical algorithms to encode information. The secret is obtained through a secret key which is used as a seed for the algorithms. The computational difficulty of the algorithms, the absence of a secret key, etc., make it impossible for an attacker to obtain the original information even if he knows the algorithm used for the encryption.
Requires the entire cryptographic system to communicate confidentially.	Modern cryptography requires that parties interested in secure communication have only the secret key.

Cryptographic context:

Cryptology, the study of cryptosystems, can be divided into two branches:

- Encryption
- Cryptoanalysis



What is encryption?:

Cryptography is the art and science of creating a cryptographic system that can provide information security.

Encryption deals with the effective protection of digital data. It refers to the design of mechanisms based on mathematical algorithms that provide fundamental services for information security. You can think of cryptography as creating a large set of tools that contains different techniques in security applications.

What is cryptanalysis?:

The art and science of decrypting ciphertext is known as cryptanalysis.

Cryptography is the twin branch of cryptography, and the two coexist. The cryptographic process outputs the ciphertext for transmission or storage. It involves studying cryptographic mechanisms with the intention of breaking them. Cryptography is also used when designing new cryptographic techniques to test their security strengths.

Note: Cryptography refers to the design of cryptographic systems, while cryptanalysis studies the violation of cryptographic systems.

Cryptographic Security Services:

The primary purpose of using cryptography is to provide the following four basic services for information security. Let's now take a look at the possible goals that cryptocurrencies intend to achieve.

Confidentiality:

Confidentiality is the fundamental security service provided by cryptography. It is a security service that keeps the information of an unauthorized person. Sometimes it is called privacy or secrecy.
Confidentiality can be achieved through a number of means, from physical protection to the use of mathematical algorithms for data encryption.

Data integrity:

It is a security service that takes care of identifying any data alteration. Data can be changed intentionally or accidentally by an unauthorized entity. The integrity service confirms whether or not the data is intact since it was last created, transmitted or stored by an authorized user.

Data integrity cannot prevent data from being altered, but it does provide a means of detecting if data has been tampered with in an unauthorized manner.

Authentication:

Authentication provides sender identification. Confirms to the recipient that the received data was sent only by an identified and verified sender.

The authentication service has two variants:

- Message authentication identifies the sender of the message regardless of the router or system that sent the message.
- Entity authentication is the assurance that data has been received from a specific entity, such as a particular website.

In addition to the originator, authentication can also provide security on other data-related parameters, such as creation / transmission date and time.

I do not repudiate:

It is a security service that ensures that an entity cannot refuse ownership of a previous commitment or action. It is a guarantee that the original creator of the data cannot deny the creation or transmission of such data to a recipient or to a third party.

Non-repudiation is a property that is most desirable in situations where the possibility of a data exchange dispute exists. For example, once an order has been placed electronically, a buyer cannot reject the purchase order if the non-repudiation service has been enabled in this transaction.

Cryptographic primitives:

Cryptographic primitives are nothing more than cryptographic tools and techniques that can be selectively used to provide a desired set of security services.

- Encryption
- Hash functions
- Message Authentication Codes (MAC)
- Digital signatures

The following table shows the primitives that can create a particular security service on their own.

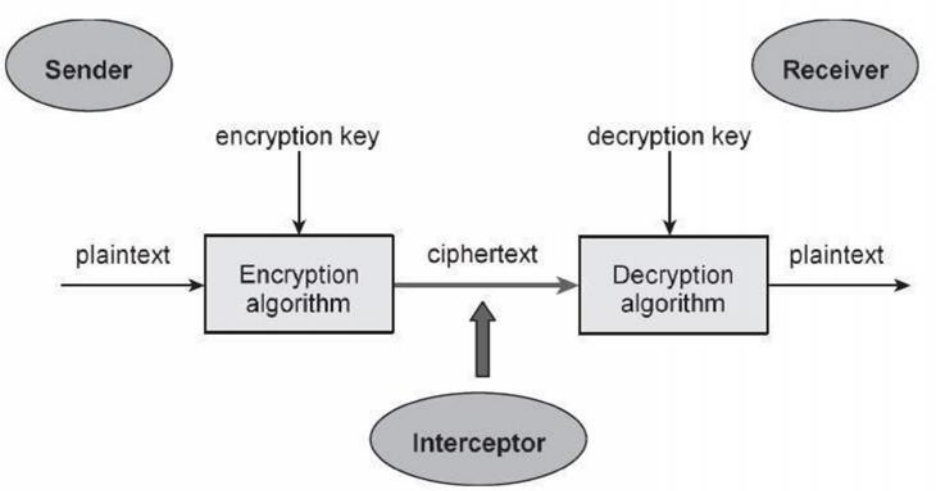
Primitives ⇨ Service ⇩	Encryption	Hash Function	MAC	Digital Signature
Confidentiality	Yes	No	No	No
Integrity	No	Sometimes	Yes	Yes
Authentication	No	No	Yes	Yes
Non Reputation	No	No	Sometimes	Yes

Note: Cryptographic primitives are closely related and are often combined to obtain a desired set of security services from a cryptosystem.

Cryptosystems:

A cryptosystem is an implementation of cryptographic techniques and the infrastructure that accompanies them to provide information security services. A cryptographic system is also known as an encryption system.

Let's analyze a simple cryptosystem model that provides confidentiality to the information transmitted. This basic model is shown in the following illustration:



The illustration shows a sender who wants to transfer some confidential data to a recipient in such a way that any interlocutor or interception on the communication channel cannot extract the data.

The goal of this simple cryptographic system is that at the end of the process, only the sender and recipient know the plaintext.

Components of a cryptosystem:

The various components of a basic cryptosystem are as follows:

- Plain text. These are the data to be protected during transmission.
- Encryption algorithm. It is a mathematical process that produces ciphertext for any plaintext and encryption key. It is a cryptographic algorithm that accepts plain text and an encryption key as input and produces encrypted text.
- Encrypted text. It is the encrypted version of the plaintext produced by the encryption algorithm that uses a specific encryption key. The ciphertext is not protected. Scrolls in public channels. It can be intercepted or compromised by anyone with access to the communication channel.
- The decryption algorithm is a mathematical process that produces a unique plaintext for any ciphertext and decryption key. It is a cryptographic algorithm that accepts a ciphertext and a decryption key

as input and generates plain text. The decryption algorithm basically reverses the encryption algorithm and is therefore closely related to it.

- Encryption key. It is a value known to the sender. The sender enters the encryption key into the encryption algorithm along with the plaintext to compute the ciphertext.
- Decryption key. It is a value known to the recipient. The decryption key is related to the encryption key, but it is not always identical to it. The recipient enters the decryption key into the decryption algorithm along with the ciphertext to calculate the plaintext.

For a given cryptographic system, a collection of all possible decryption keys is called the key space.

An interceptor (an attacker) is an unauthorized entity that attempts to determine plain text. You can view the ciphertext and learn about the decryption algorithm. However, you should never know the decryption key.

Types of cryptosystems:

Basically, there are two types of cryptosystems based on the way encryption-decryption is performed on the system:

- Symmetric key encryption
- Asymmetric key encryption

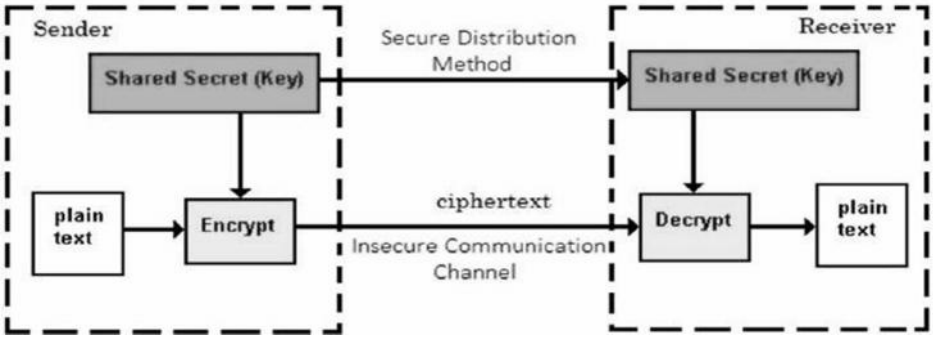
The main difference between these cryptographic systems is the relationship between the encryption and the decryption key. Logically, in any cryptographic system, both keys are closely associated. It is practically impossible to decrypt the ciphertext with the key that is not related to the encryption key.

Symmetric key encryption:

The encryption process in which the same keys are used to encrypt and decrypt information is known as symmetric key encryption.

The study of symmetric cryptosystems is known as symmetric cryptography. Symmetric cryptosystems are sometimes also called secret key cryptosystems.

Some known examples of symmetric key encryption methods are: Digital Encryption Standard (DES), Triple-DES (3DES), IDEA, and BLOWFISH.



Before 1970, all cryptographic systems used symmetric key cryptography. Even today, its relevance is very high and it is widely used in many cryptographic systems. This encryption is highly unlikely to disappear as it has some advantages over asymmetric key encryption.

The most important features of the cryptosystem based on symmetric key cryptography are:

- People who use symmetric key cryptography must share a common key before exchanging information.
- It is recommended to change the keys regularly to avoid any attack on the system.
- A robust mechanism is needed for key exchange between communicating parties. Since the keys have to be changed regularly, this mechanism becomes expensive and cumbersome.
- In a group of n people, to allow communication between any two people, the number of keys required for the group is $n \times (n - 1) / 2$.
- The key length (number of bits) in this encryption is shorter and therefore the encryption-decryption process is faster than asymmetric key encryption.
- The processing power of a computer system required to run a symmetric algorithm is less.

Challenge of the symmetric key cryptography system:

There are two restrictive challenges in using symmetric key cryptography.

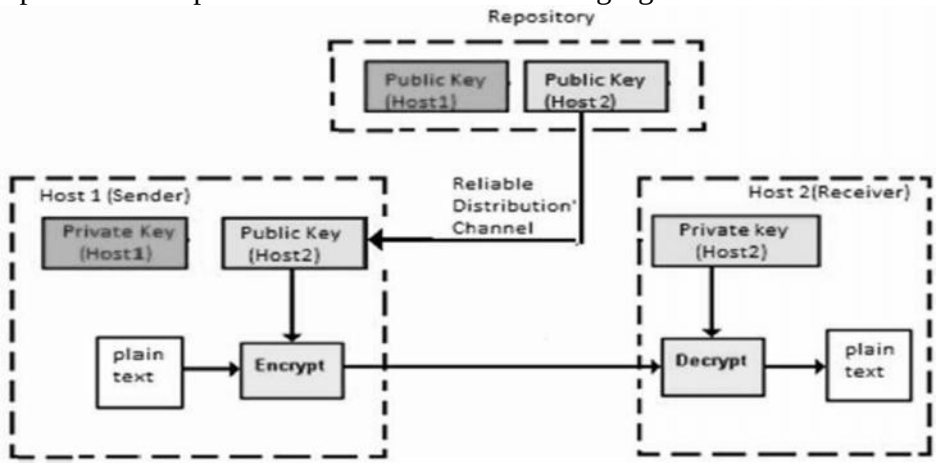
- **Establishment of the key:** Before any communication, both the sender and the recipient must agree on a secret symmetric key. Requires a secure key creation mechanism.
- **Trust issue:** Since the sender and recipient use the same symmetric key, there is an implicit requirement that the sender and recipient "trust" each other. For example, it may happen that the recipient has lost the key of an attacker and the sender is not informed.

These two challenges are very restrictive for modern communication. Today, people need to exchange information with unknown and unreliable

parties. For example, the communication between online sellers and customers. These limitations of symmetric key cryptography have led to asymmetric key cryptographic schemes.

Asymmetric key encryption:

The encryption process in which different keys are used to encrypt and decrypt information is known as asymmetric key cryptography. Although the keys are different, they are mathematically related and therefore it is possible to recover the plaintext by decrypting the ciphertext. The process is shown in the following figure:



Asymmetric key cryptography was invented in the 20th century to overcome the need for secret keys previously shared between communicating people. The most important features of this encryption scheme are as follows:

- All users of this system must have a different key pair, private key, and public key. These keys are mathematically related: when one key is used for encryption, the other can decrypt the ciphertext back to its original plaintext.
- It requires you to put the public key in a public repository and the private key as a well-kept secret. Therefore, this encryption scheme is also called public key cryptography.
- Although the user's public and private keys are related, it is not computationally possible to find each other. This is one of the strengths of this scheme.
- When Host1 needs to send data to Host2, it obtains Host2's public key from the repository, encrypts the data, and transmits.
- Host2 uses your private key to extract plain text.
- The length of the keys (number of bits) in this encryption is large and therefore the encryption-decryption process is slower than symmetric key encryption.

- The processing power of a computer system required to run an asymmetric algorithm is greater.

Symmetric cryptosystems are a natural concept. Conversely, public-key cryptosystems are rather difficult to understand.

You may be thinking, how can the encryption key and the decryption key be "related" and yet is it impossible to determine the decryption key from the encryption key? The answer is in the mathematical concepts. It is possible to design a cryptosystem whose keys have this property. The concept of public key cryptography is relatively new. Fewer public key algorithms are known than symmetric algorithms.

Challenge of the public key cryptographic system:

Public-key cryptographic systems have a significant challenge: the user must trust that the public key they are using in communications with a person is actually that person's public key and has not been forged by malicious third parties.

This is typically accomplished through a public key infrastructure (PKI) made up of a trusted third party. The third party securely manages and guarantees the authenticity of the public keys. When the third party is asked to provide the public key for any communicating person X, they rely on them to provide the correct public key.

The third party is satisfied with the user's identity through the attestation process, notarization, or some other process: whether X is the only, or globally unique, X. The most common method of making verified public keys available is to embed them in a certificate digitally signed by a trusted third party.

Relationship between encryption schemes:

Here is a summary of the basic key properties of two types of cryptosystems:

	Symmetric cryptosystems	**Public key cryptosystems**
Relationship between the keys	Same	Different, but mathematically related
Encryption key	Symmetrical	Public
Decryption key	Symmetrical	Private

Due to the advantages and disadvantages of both systems, symmetric key and public key cryptographic systems are often used together in practical information security systems.

Kerckhoff's principle for the cryptosystem:

In the 19th century, a Dutch cryptographer A. Kerckhoff provided the requirements for a good cryptographic system. Kerckhoff said that a cryptographic system should be secure even if everything about the system, except the key, is in the public domain. The six design principles defined by Kerckhoff for the cryptosystem are:

- The cryptosystem should be practically indestructible, if not mathematically.
- The fall of the cryptosystem into the hands of an intruder must not involve any compromise of the system, avoiding inconvenience for the user.
- The key must be easily communicable, memorable and modifiable.
- The ciphertext must be transmissible by telegraph, an insecure channel.
- The encryption device and documents must be portable and managed by one person.
- Finally, the system must be easy to use, requiring no mental effort or knowledge of a long set of rules to follow.

The second rule is known today as the Kerckhoff principle. It is applied practically in all contemporary encryption algorithms such as DES, AES, etc. These public algorithms are considered completely safe. The security of the encrypted message depends solely on the security of the secret encryption key.

Keeping algorithms secret can be a significant barrier to cryptanalysis. However, keeping algorithms secret is only possible if they are used in a strictly limited circle.

In the modern era, cryptocurrencies must satisfy internet-connected users. In such cases, the use of a secret algorithm is not feasible, so Kerckhoff's principles have become essential guidelines for algorithm design in modern cryptography.

Attacks on the cryptosystem:

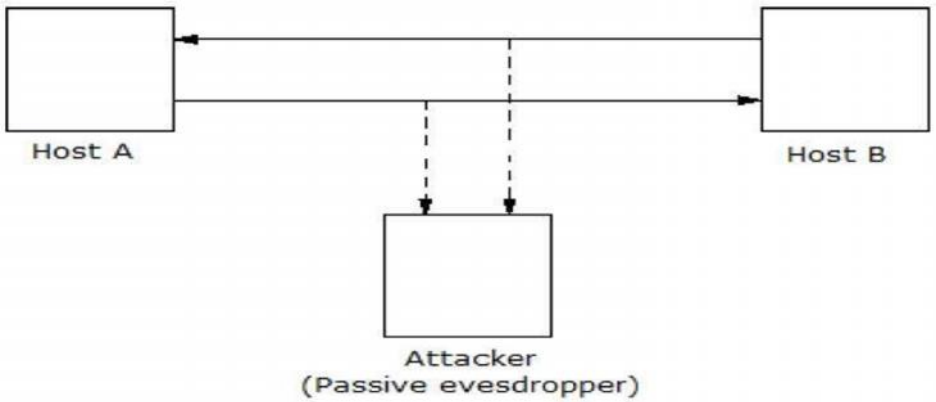
In today's era, not just business, but almost every aspect of human life is information-driven. Therefore, it has become imperative to protect useful information from malicious activities such as attacks. Let's consider the types of attacks that information is often subjected to.

Attacks are generally classified according to the action taken by the attacker. An attack, therefore, can be passive or active.

Passive attacks:

The main goal of a passive attack is to gain unauthorized access to information. For example, actions such as interception and interception on the communication channel can be considered passive attacks.

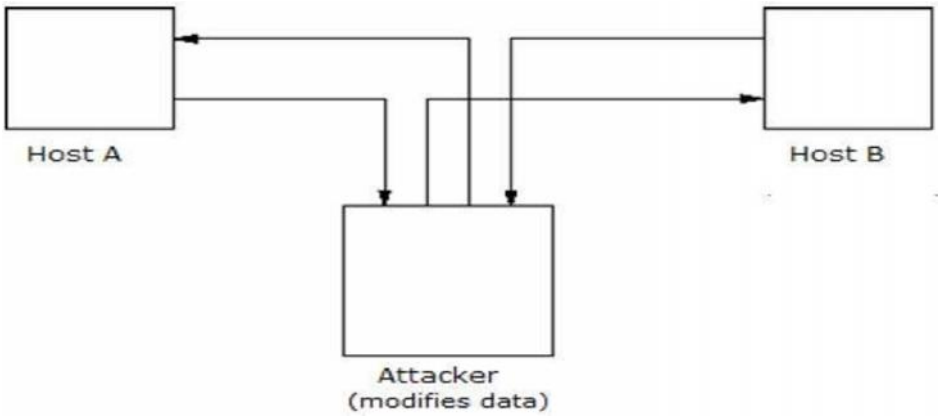
These actions are passive in nature, as they do not affect information or disturb the communication channel. A passive attack is often seen as information theft. The only difference between physical asset theft and information theft is that data theft still leaves the owner in possession of that data. Passive information attack is therefore more dangerous than property theft, as information theft can go unnoticed by the owner.



Active attacks:

An active attack involves modifying information in some way by executing a process on the information. For example,

- Change information in an unauthorized way.
- Initiate an unintended or unauthorized transmission of information.
- Alteration of authentication data, such as the sender name or timestamp associated with the information
- Unauthorized deletion of data.
- Denial of access to information for legitimate users (denial of service).



Cryptography provides many tools and techniques for implementing cryptographic systems that can prevent most of the attacks described above.

Hypothesis of the attacker:

Let's take a look at the prevailing environment around cryptosystems, followed by the types of attacks used to breach these systems:

Environment around the cryptosystem:

When considering possible attacks on the cryptosystem, it is necessary to understand the environment of the cryptosystem. The assumptions of the attacker and the knowledge of the environment determine his abilities.

In cryptography, the following three assumptions are made about the security environment and the attacker's capabilities.

Details of the encryption scheme:

The design of a cryptographic system is based on the following two cryptographic algorithms:

- **Public algorithms:** with this option all the details of the algorithm are in the public domain, known to all.
- **Proprietary Algorithms:** The details of the algorithm are known only to system designers and users.

In the case of proprietary algorithms, security is guaranteed by obscurity. Private algorithms may not be the most powerful as they are developed in-house and may not be thoroughly researched to identify weaknesses.

Secondly, they only allow communication between closed groups. Therefore, they are not suitable for modern communication where people communicate with a large number of known or unknown entities. Also,

according to Kerckhoff's principle, it is preferable that the algorithm be public with the strength of the encryption in the key.

Therefore, the first assumption about the security environment is that the attacker knows the encryption algorithm.

Availability of the ciphertext:

We know that once the plaintext is encrypted into ciphertext, it is placed on an insecure public channel (e.g. email) for transmission. Therefore, the attacker can obviously assume that he has access to the ciphertext generated by the cryptosystem.

Availability of plain and encrypted text:

This hypothesis is not as obvious as the others. However, there could be situations where an attacker could gain access to the plaintext and the corresponding ciphertext. Some of these possible circumstances are:

- The attacker tricks the sender to convert the plaintext of his choice and gets the ciphertext.
- The recipient may inadvertently disclose the plaintext to the attacker. The attacker has access to the corresponding ciphertext collected from the open channel.
- In a public key cryptographic system, the cryptographic key is in the open domain and is known to any potential attacker. With this key it is possible to generate pairs of plaintext and corresponding ciphertext.

Cryptographic attacks:

The basic intention of an attacker is to crack a cryptographic system and find the plaintext of the ciphertext. To get the plaintext, the attacker only needs to find out the secret decryption key, as the algorithm is already in the public domain.

Therefore, apply the utmost effort to find out the secret key used in the cryptosystem. Once the attacker can determine the key, the attacked system is considered broken or compromised.

Based on the methodology used, attacks on cryptographic systems are classified as follows:

- **Ciphertext Only (COA) Attacks:** In this method, the attacker has access to a series of encrypted texts. You do not have access to the corresponding plaintext. The COA is said to be successful when the corresponding plaintext can be determined from a given set of ciphertext. Occasionally, the encryption key can be determined by this attack. Modern cryptographic systems are protected from attacks based only on ciphertext.

- **Known plain text attack (KPA):** In this method, the attacker knows the plaintext of some parts of the ciphertext. The task is to decrypt the rest of the ciphertext using this information. This can be done by determining the key or by some other method. The best example of this attack is linear cryptanalysis against block ciphers.
- **Chosen plain text attack (CPA)**: In this method, the attacker has encrypted the text of his choice. Then you have the ciphertext-plaintext pair of your choice. This simplifies the task of determining the encryption key. An example of this attack is differential cryptanalysis applied against block ciphers and hash functions. RSA, a popular public key cryptographic system, is also vulnerable to chosen plaintext attacks.
- **Attack dictionary:** This attack has many variations, all of which involve compiling a 'dictionary'. In the simplest method of this attack, the attacker constructs a ciphertext dictionary and the corresponding plaintext that he has learned over time. In the future, when an attacker obtains the ciphertext, they refer to the dictionary to find the corresponding plaintext.
- **Brute Force Attack (BFA):** In this method, the attacker tries to determine the key by trying all possible keys. If the key is 8 bits long, the number of possible keys is 28 = 256. The attacker knows the ciphertext and the algorithm, now he tries all 256 keys one by one to decrypt them. The time to complete the attack would be very long if the key is long.
- **Birthday Attack:** This attack is a variation of the brute force technique. It is used against the cryptographic hash function. When students in a class are asked about their birthdays, the answer is one of 365 possible dates. Suppose the first student's date of birth is August 3. So, to find the next student whose birthday is August 3, we need to ask $1.25 * \sqrt{365} \approx 25$ students. Likewise, if the hash function produces 64-bit hash values, the possible hash values are 1.8x1019. By repeatedly evaluating the function for different inputs, the same output is expected to be obtained after approximately 5.1x109 random inputs. If the attacker can find two different entries that give the same hash value,
- **Medium Attack Man (MIM):** The targets of this attack are mostly public-key cryptographic systems where key exchange occurs before communication takes place.
    - Host A wants to communicate with host B, so it requests B's public key.
    - An attacker intercepts this request and sends his public key instead.
    - Therefore, any host A sends to host B, the attacker can read.
    - To maintain communication, the attacker re-encrypts the data after reading it with his public key and sends it to B.

- The attacker sends his public key as A's public key for B to take as if he were taking it from A.

- **Side channel connection (SCA):** This type of attack is not against any particular algorithm or cryptographic system. Instead, it is launched to exploit the weakness in the physical implementation of the cryptosystem.
- **Time Attacks:** They take advantage of the fact that different calculations take different times to be calculated in the processor. By measuring these times it is possible to know a particular calculation that the processor is making. For example, if encryption takes longer, it indicates that the secret key is long.
- **Power analysis attacks:** These attacks are similar to time attacks, except that the amount of power consumption is used to gain insight into the nature of the underlying calculations.
- **Failure Analysis Attacks:** In these attacks, errors are induced in the cryptosystem and the attacker studies the resulting output to obtain useful information.

Practicality of attack:

The attacks on cryptographic systems described here are highly academic, as most of them come from the academic community. Indeed, many academic attacks involve unrealistic assumptions about the attacker's environment and abilities. For example, in a chosen ciphertext attack, the attacker requests an impractical number of deliberately chosen plaintext-text pairs. It may not be entirely practical.

However, the fact that there is an attack should be cause for concern, especially if the attacking technique has the potential to improve.

1.10 IMPLEMENTING AFFINE CRYPTOGRAPHY

Affine cipher is a type of mono-alphabetic substitution cipher, in which each letter of an alphabet is mapped to its numerical equivalent, encrypted using a simple mathematical function, and converted back into a letter. The formula used means that each letter is encrypted into another letter and vice versa, which means that the encryption is essentially a standard replacement encryption with a rule governing which letter goes to which.

The whole process is based on working in modulo m (the length of the alphabet used). In affine cipher, the letters of an alphabet of size m are first assigned to integers in the range 0… m-1.

The "key" for affine encryption is composed of 2 numbers, we will call them a and b. The following discussion assumes the use of a 26-

character alphabet (m = 26). a must be chosen to be a relative prime m (that is, a must have no factors in common with m).

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

Encryption:

Use modular arithmetic to transform the integer that each plaintext letter corresponds to into another integer that matches a ciphertext letter. The encryption function for a single letter is

E (x) = (ax + b) mod m

module m: size of the alphabet

a and b: encryption key.

a must be chosen so that a and m are coprime.

decoded

When decrypting the ciphertext, you need to perform the opposite (or reverse) functions on the ciphertext to recover the plaintext. Again, the first step is to convert each letter in the ciphertext to its integer values. The decryption function is

D (x) = a ^ -1 (x - b) mod m

a ^ -1: inverse modular multiplicative of a modulo of my, satisfies the equation

1 = aa ^ -1 mod m.

To find a multiplicative inverse:

We need to find a number x such that:

If we find the number x such that the equation is true, then x is the inverse of a, and we call it a ^ -1. The easiest way to solve this equation is to look up each of the numbers 1 through 25 and see which one satisfies the equation.

[g, x, d] = gcd (a, m); % we can ignore ged, we don't need them

x = mod (x, m);

If you now multiply x and ay reduce the result (mod 26), you will get the answer 1. Remember, this is only the definition of an inverse, that is, if a * x = 1 (mod 26), then x is an inverse of a (ya is inverse of x)

Example:

Encryption: Key Values a=17, b=20

Original Text	T	W	E	N	T	Y		F	I	F	T	E	E	N
x	19	22	4	13	19	24		5	8	5	19	4	4	13
ax+b % 26*	5	4	10	7	5	12		1	0	1	5	10	10	7
Encrypted Text	F	E	K	H	F	M		B	A	B	F	K	K	H

Decryption: a^-1 = 23

Encrypted Text	F	E	K	H	F	M		B	A	B	F	K	K	H
Encrypted Value	5	4	10	7	5	12		1	0	1	5	10	10	7
23 *(x-b) mod 26	19	22	4	13	19	24		5	8	5	19	4	4	13
Decrypted Text	T	W	E	N	T	Y		F	I	F	T	E	E	N

1.11 MONOALPHABETIC AND POLYALPHABETIC CRYPTOGRAPHY

The monoalphabetic cipher is a substitution cipher in which, for a given key, the cipher alphabet of each simple alphabet is fixed during the encryption process. For example, if 'A' is encrypted as 'D', for any number of occurrences in that plain text, 'A' will always be encrypted as 'D'.

All the substitution ciphers we discussed earlier in this chapter are mono-alphabetic; these ciphers are very susceptible to cryptanalysis.

The polyalphabetic cipher is a substitution cipher in which the ciphered alphabet of the simple alphabet can be different at different points during the encryption process. The next two examples, playfair and Vigenere Cipher, are polyalphabetic ciphers.

Playfair encryption:

In this scheme, pairs of letters are encrypted, rather than single letters, as is the case with simple surrogate encryption.

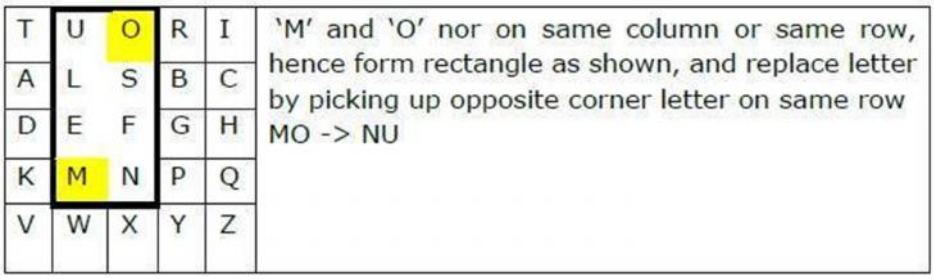
In playfair encryption, a key table is initially created. The key table is a 5 × 5 alphabet grid that serves as a key to encrypt the plaintext. Each of the 25 alphabets must be unique and one letter of the alphabet (usually J) is omitted from the table, as we only need 25 alphabets instead of 26. If the plaintext contains J, it is replaced by I.

Sender and recipient decide on a particular key, let's say "tutorial". In a key table, the first characters (left to right) in the table are the sentence, excluding duplicate letters. The rest of the table will be filled with the remaining letters of the alphabet, in natural order. The key table appears to be:

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

Playfair encryption process:

- First, a plain text message is divided into two-letter pairs (digraphs). If the number of letters is odd, a Z is added to the last letter. Suppose we want to encrypt the "hide money" message. It will be written as −HI DE MO NE YZ
- The encryption rules are:
  - If both letters are in the same column, take the letter below each (go back to the top if it is below)
- "H" and "I" are in the same column, so take the letter below them to replace them. HI → QCI If both letters are in the same row, take the letter to the right of each (returning to the left if it is further to the right)
  "D" and "E" are in the same row, so take the letter to the right of them to replace them. DE → EFI If neither of the above two rules are true, form a rectangle with the two letters and take the letters at the opposite horizontal corner of the rectangle.



Using these rules, the result of "hide money" encryption with the "tutorial" key would be:

QC EF NU MF ZV

Cracking Playfair's encryption is as simple as doing the same process in reverse. The recipient has the same key and can create the same key table, then decrypt any message created with that key.

Security value:

It is also a surrogate cipher and is difficult to crack compared to simple surrogate cipher. As in the case of the surrogate cipher,

cryptanalysis is also possible in the Playfair cipher, however it would be against 625 possible pairs of letters (25x25 alphabets) instead of 26 different possible alphabets.

Playfair's encryption was mainly used to protect important, but not critical, secrets, as it is quick to use and requires no special equipment.

1.12 CRYPTOANALYSIS OF THE VIGENERE CIPHER

This encryption scheme uses a text string (such as a word) as a key, which is then used to make a number of changes to the plain text.

For example, suppose the key is "period". Each alphabet on the key is converted to its respective numerical value: In this case,

p → 16, o → 15, i → 9, n → 14 and t → 20.

Therefore, the key is: 16 15 9 14 20.

Vigenere encryption process:

- The sender and recipient decide on a key. Saying "point" is the key. The numeric representation of this key is '16 15 9 14 20 '.
- The sender wants to encrypt the message, for example "attack from the southeast". You will arrange the plain text and number keys as follows:

a	t	t	a	c	k	f	r	o	m	s	o	u	t	h	e	a	s	t
16	15	9	14	20	16	15	9	14	20	16	15	9	14	20	16	15	9	14

- Now change each plaintext alphabet to the number written below to create a ciphertext as shown below:

a	t	t	a	c	k	f	r	o	m	s	o	u	t	h	e	a	s	t
16	15	9	14	20	16	15	9	14	20	16	15	9	14	20	16	15	9	14
Q	I	C	O	W	A	U	A	C	G	I	D	D	H	B	U	P	B	H

- Here, each plain text character has been changed by a different amount and that amount is determined by the key. The key must be less than or equal to the size of the message.
- For decryption, the recipient uses the same key and changes the received ciphertext in reverse order to get the plaintext.

Q	I	C	O	W	A	U	A	C	G	I	D	D	H	B	U	P	B	H
16	15	9	14	20	16	15	9	14	20	16	15	9	14	20	16	15	9	14
a	t	t	a	c	k	f	r	o	m	s	o	u	t	h	e	a	s	t

Security value:

Vigenere Cipher was designed by modifying the standard Caesar cryptography to reduce the effectiveness of cryptanalysis on ciphertext and make a cryptosystem more robust. It is significantly more secure than regular Caesar encryption.

Throughout history, it has been used regularly to protect sensitive political and military information. It became known as indestructible cryptography due to the difficulty it represented for cryptanalysis.

Variants of Vigenere encryption:

There are two special cases of Vigenere encryption:

- The keyword length is the same as for a plain text message. This case is called Vernam Cipher. It is more secure than typical Vigenere encryption.
- Vigenere cryptography becomes a cryptosystem with perfect secrecy, which is called One-Time Notepad.

One-time pillow:

Circumstances are ...

- The length of the keyword is equal to the length of the normal text.
- The keyword is a randomly generated string of alphabets.
- The keyword is used only once.

Security value:

Let's compare a Shift cipher with a one-time pad.

Exchange encryption - easy to crack

In the case of Shift encryption, the entire message could have shifted between 1 and 25. This is a very small size and very easy to use by brute force. However, now that each character has their own individual change between 1 and 26, the possible keys grow exponentially for the message.

Disposable pad - impossible to break:

Suppose you encrypt the name "period" with a single-use notepad. It is a 5 letter text. To decrypt the ciphertext by brute force, you need to test all the key possibilities and perform the calculation for (26 x 26 x 26 x 26 x 26) = 265 = 11881376 times. This is for a message with 5 alphabets. Therefore, for a longer message, the calculation grows exponentially with

each additional alphabet. This makes it computationally impossible to break ciphertext by brute force.

1.13 LINEAR FEEDBACK SHIFT REGISTER FLOW CIPHERS (LFSR)

- A linear feedback shift register (LFSR) is a type of digital circuit that has several storage areas, each of which can contain 1 bit, linked in a chain. The output of each storage area is connected to the input of the next storage area in the chain, resulting in a circuit that shifts the data stored in it one position to the right each time the circuit is executed. How storage areas are connected varies from circuit to circuit, and each setting will change the pattern in which bits move from one storage area to another.
- LFSRs have many important uses in digital communication, not just cryptography. They are used in TV broadcast signals, data transfer via USB cable, and GPS navigation. LFSR circuits are still used to encrypt GSM cell phone signals, despite serious security concerns.

LFSR stream encryption:

We encrypt the C3P message (abbreviated from C3P0) using a 4-bit LFSR with seed 0110 and definition:

b4 ← b′1 + b′2 + b′4

b4 ← b1 ′ + b2 ′ + b4 ′

The first step is to convert the ASCII string to binary:

Plain ASCII text: C3P

plain text binary: 01000011 00110011 01010000

Then, it calculates the output stream of the LFSR so that there are as many bits as are needed for the message. In this case 24.

B. 4 b4	B. 3 b3	B. Two b2	B. 1 b1
0	1	1	0
1	0	1	1
1	1	0	1

0	1	1	0
1	0	1	1
1	1	0	1
0	1	1	0
1	0	1	1
1	1	0	1
0	1	1	0
1	0	1	1
1	1	0	1
0	1	1	0
1	0	1	1
1	1	0	1
0	1	1	0
1	0	1	1
1	1	0	1
0	1	1	0
1	0	1	1
1	1	0	1
0	1	1	0
1	0	1	1

1	1	0	1

Looking at the far right column, you get the bitstream:
01101101 10110110 11011011

Performing XOR encryption on these key streams:
01000011 00110011 01010000
01101101 10110110 11011011
00101110 10000101 10001011
01000011 00110011 01010000⊕ 01101101 10110110 11011011001011100000101 10001011
Which once converted to Base64 produces LoWL

To decrypt an LFSR stream cipher, you must bitwise subtract the key stream from the ciphertext binary stream. Fortunately, bitwise subtraction is identical to bitwise addition, so you just need to use the XOR operation again.

Base64 ciphertext: LoWL
binary ciphertext: 001011 101000 010110 001011

Using the same binary keystream: 011011 011011 011011 011011
001011 101000 010110 001011
011011 011011 011011 011011
010000 110011 001101 010000
001011 101000 010110 001011⊕ 011011 011011 011011 011011010000 110011 001101 010000
Which is converted back to plain ASCII text: C3P

1.14 SHANNON'S THEORY

Information is the source of a communication system, be it analog or digital. Information theory is a mathematical approach to the study of information encoding along with the quantification, storage and communication of information.

Conditions of occurrence of events:

If we consider an event, there are three conditions of occurrence.

- If the event has not occurred, there is a condition of uncertainty.
- If the event has just occurred, there is a surprise condition.

- If the event occurred some time ago, there is a condition to have some information.

Therefore, these three occur at different times. The difference in these conditions helps us to have a knowledge of the probabilities of occurrence of events.

Entropy:

When we look at the probabilities of an event occurring, whether it is surprising or uncertain, it means we are trying to get an idea of the average content of information from the source of the event.

Entropy can be defined as a measure of the average information content per character symbol. Claude Shannon, the "father of information theory", gave you a formula like

$$ H = - \ sum_ {i} p_i \ log_ {b} p_i $$

Where $ p_i $ is the probability that character number i of a given stream of characters will occur and b is the basis of the algorithm used. Hence, this is also called Shannon entropy.

The amount of uncertainty that remains on the channel input after observing the channel output is called conditional entropy. It is denoted by $ H (x \ arrowvert y) $

Discreet character with no memory:

A source from which data is output at successive intervals, which is independent of previous values, can be called a discrete and memoryless source.

This source is discrete since it is not considered for a continuous time interval, but in discrete time intervals. This character has no memory, as it is always up to date, regardless of previous values.

Source encoding:

According to the definition, "Given a memoryless discrete entropy source $ H (\ delta) $, the average code length $ \ bar {L} $ for any source encoding is limited to $ \ bar {L} \ geq H (\ delta) $ ".

In simpler words, the code word (eg the Morse code for the word QUEUE is -.- ..-. ..-.) Is always greater than or equal to the source code (QUEUE in the example). This means that the symbols in the code are greater than or equal to the alphabets in the source code.

Channel coding:

Coding the channels in a communication system introduces redundancy with a control, to improve the reliability of the system. Source coding reduces redundancy to improve system efficiency.

Channel coding consists of two action parts.

- Mapping the incoming data stream to a channel input stream.
- Invert the mapping of the channel's output sequence to an output data sequence.

The ultimate goal is to minimize the overall effect of channel noise. Mapping is done by the transmitter, with the help of an encoder, while reverse mapping is done by the receiver via a decoder.

1.15 CRYPTOGRAPHIC SECURITY SERVICES

The primary purpose of using cryptography is to provide the following four basic services for information security. Let's now take a look at the possible goals that cryptocurrencies intend to achieve.

Confidentiality:

- Confidentiality is the fundamental security service provided by cryptography. It is a security service that keeps the information of an unauthorized person. Sometimes it is called privacy or secrecy.
- Confidentiality can be achieved through a number of means, from physical protection to the use of mathematical algorithms for data encryption.

Data integrity:

- It is a security service that takes care of identifying any data alteration. Data can be changed intentionally or accidentally by an unauthorized entity. The integrity service confirms whether or not the data is intact since it was last created, transmitted or stored by an authorized user.
- Data integrity cannot prevent data from being altered, but it does provide a means of detecting if data has been tampered with in an unauthorized manner.

Authentication:

Authentication provides sender identification. Confirms to the recipient that the received data was sent only by an identified and verified sender.

The authentication service has two variants:

- Message authentication identifies the sender of the message regardless of the router or system that sent the message.
- Entity authentication is the assurance that data has been received from a specific entity, such as a particular website.

In addition to the originator, authentication can also provide security on other data-related parameters, such as creation / transmission date and time.

I do not repudiate:

- It is a security service that ensures that an entity cannot refuse ownership of a previous commitment or action. It is a guarantee that the original creator of the data cannot deny the creation or transmission of such data to a recipient or to a third party.
- Non-repudiation is a property that is most desirable in situations where the possibility of a data exchange dispute exists. For example, once an order has been placed electronically, a buyer cannot reject the purchase order if the non-repudiation service has been enabled in this transaction.

Cryptographic primitives:

Cryptographic primitives are nothing more than cryptographic tools and techniques that can be selectively used to provide a desired set of security services.

- Encryption
- Hash functions
- Message Authentication Codes (MAC)
- Digital signatures

The following table shows the primitives that can create a particular security service on their own.

Primitives ⇨ Service ⇩	Encryption	Hash Function	MAC	Digital Signature
Confidentiality	Yes	No	No	No
Integrity	No	Sometimes	Yes	Yes
Authentication	No	No	Yes	Yes
Non Reputation	No	No	Sometimes	Yes

Note: Cryptographic primitives are closely related and are often combined to obtain a desired set of security services from a cryptosystem.

1.16 HUFFMANMAN CODE

- A Huffman code is defined as a particular type of optimal prefix code commonly used for lossless data compression.
- The process of finding or implementing such code proceeds through Huffman coding, an algorithm developed by David A. Huffman when he was a Sc.D. student at MIT and published in 1952 in the article "A method for the construction of minimum redundancy codes".
- The output of the Huffman algorithm can be viewed as a variable-length code page to encode a source symbol (such as a character in a file). The algorithm creates this table from the estimated probability or frequency of occurrence (weight) for each possible value of the source symbol. As with other entropy encoding methods, more common symbols are generally represented by implementing fewer bits than less common symbols. Huffman's method can be implemented efficiently, finding a code in linear time with respect to the number of input weights if these weights are sorted.

Entropy:

- In information theory, Shannon's source coding theorem (or silent coding theorem) is able to establish the limits to possible data compression and the operational meaning of Shannon's entropy.
- The source code theorem shows that (at the limit, since the length of a data stream of independent and identically distributed random variables (iid) tends to infinity) it is not possible to compress the data in such a way that the code rate (number of bits averages per symbol) is less than the Shannon entropy of the source, with virtually no certainty that the information will be lost. However, it is possible to obtain the code rate arbitrarily close to Shannon's entropy, with negligible probability of loss.
- Information entropy is defined as the average rate at which information is produced from a stochastic data source.

Calculate the entropy of a random variable:

We can also calculate how much information there is in a random variable.

For example, if we wanted to compute information for a random variable X with probability distribution p, this could be written as a function H (); for example: H (X)

In fact, the calculation of the information of a random variable is similar to the calculation of the information of the probability distribution of the events of the random variable.

The computation of the information of a random variable is called "information entropy", "Shannon entropy" or simply "entropy".

It is related to the idea of entropy in physics by analogy, in the sense that both refer to the uncertainty of the term.

The intuition of entropy is that it is defined as the average number of bits needed to represent or transmit an event extracted from the probability distribution of the random variable.

The Shannon entropy of a distribution is defined as the expected amount of information in an event extracted from that distribution.
Provides a lower limit on the number of bits needed on average to encode symbols extracted from a P distribution.

Entropy can be calculated for a random variable X with k in K discrete states as follows
H (X) = -sum (each k in K p (k) * log (p (k)))

This means the negative of the sum of the probability of each event multiplied by the logarithm of the probability of each event.

For information, the log () function implements base-2 and the units are bits. Instead, a natural logarithm can be implemented.

The lowest entropy is calculated for a random variable that has a single event with a probability of 1.0, a certainty. The greatest entropy for a random variable will be possible if all events occur with the same probability.

1.17 DISTANCE OF UNIQUENESS

- Distance of uniqueness is a property of a given encryption algorithm. It answers the question "if we performed a brute force attack, how much ciphertext would we need to make sure our solution is the real one?" The answer depends on the redundancy of English. Let's hope a short example sheds light on the problem:
- Let's say, for example, that we are given a message to decrypt: FJKFPO, and we know that it is encrypted with a surrogate cipher. Can we understand it? The answer is "not really". We can find many English words that fit the pattern, sure, but we'll never know what the real original plaintext was. For example, all of the following English fragments are legitimately decrypted from the ciphertext above and we have no idea which is correct:

that means
from your
railway station

into you
our
the top
both
Oxford
what in?
the one of
and many others. The longer the ciphertext, the less possible decryption. We want to know, how long does a piece of ciphertext have to last before it has a single possible decryption? This minimum length is given by the distance of uniqueness.

Redundancy:

The replacement cipher has a 26-letter key. The total number of keys is the number of ways we can mix 26 letters or 26! (factorial), this is a large number of possible keys. The amount of information it carries, measured in bits, is given by the logarithm to base 2:

$$\log_2(26!) = 88.28$$

or about 88 bits. The amount of information (per letter) contained in a 26-letter alphabet is $\log_2(26) = 4.7$ bit. The actual amount of information carried by English turns out to be around 1.5 bits per character. This means that English redundancy is approximately 4.7 - 1.5 = 3.2 bits per character.

Distance of uniqueness:

- The distance of uniqueness is the ratio of the number of bits needed to express the key divided by the redundancy of English in bits per character. In the case of the replacement figure above, this is 88.28 / 3.2 = 27.6. This means that a minimum of 28 characters are required to ensure that a particular decryption is unique. Different ciphers have different unique distances, a greater unique distance generally indicates more secure encryption.
- The calculation of the distance of uniqueness is based on the fact that all keys are equally probable. If the key for an encryption is an English word, this severely limits the number of possible keys. Consequently, the Unicity distance will be less.

Fake keys:

Definition:

If you know that plain text is taken from a "natural" language, then knowing the ciphertext discards a certain subset of keys. Of the possible

remaining keys, only one is correct. The remaining possible but incorrect keys are called false keys.

Example:

A decryption of the turnaround cipher:

Ciphertext = WNAJW

Decryption with E - RIVER

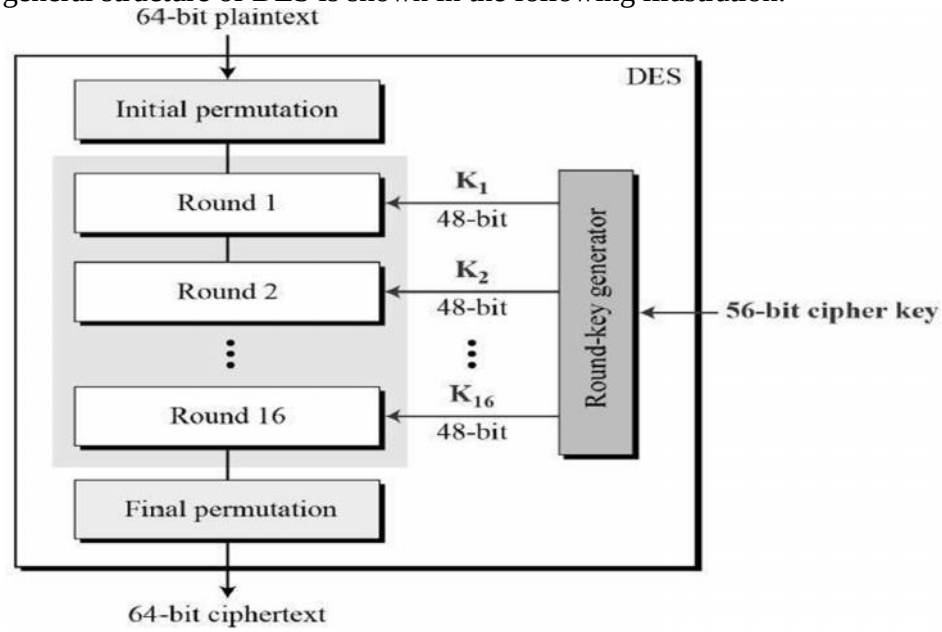
Decryption with W - ARENA

E or W is the correct key. The other is a fake password.

1.18 THE DATA ENCRYPTION STANDARD

The Data Encryption Standard (DES) is a symmetric key block cipher published by the National Institute of Standards and Technology (NIST).

DES is an implementation of a Feistel cipher. Uses 16 round Feistel frame. The block size is 64 bits. Although the key length is 64 bits, DES has an effective key length of 56 bits, as the encryption algorithm does not use 8 of the key's 64 bits (it works only as a verification bit). The general structure of DES is shown in the following illustration:

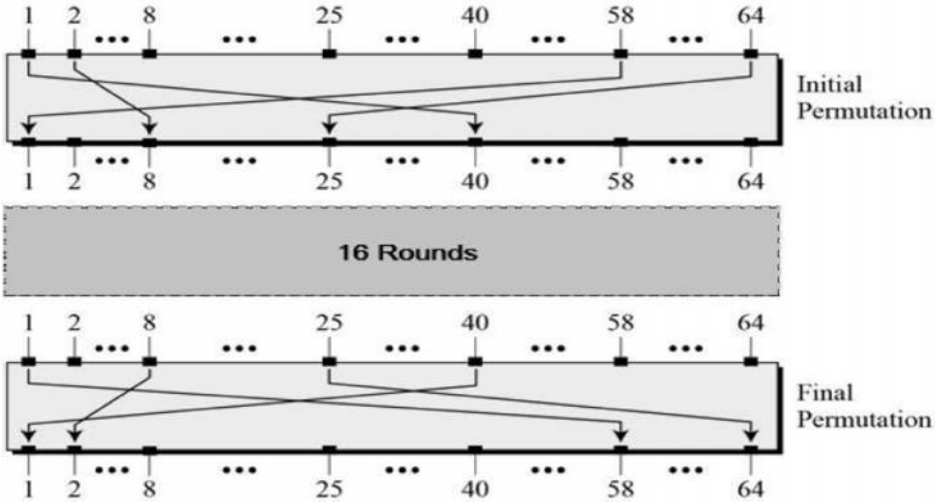


Since DES relies on Feistel encryption, all that is needed to specify DES is:

- Round feature
- Key times
- Any additional processing: initial and final permutation

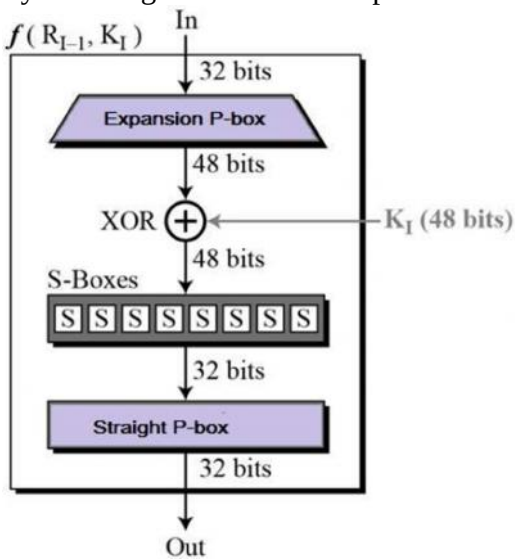
Initial and final permutation:

The initial and final permutations are straight permutation boxes (P-boxes) that are inverse of each other. They have no cryptographic meaning in DES. The initial and final permutations are shown below:

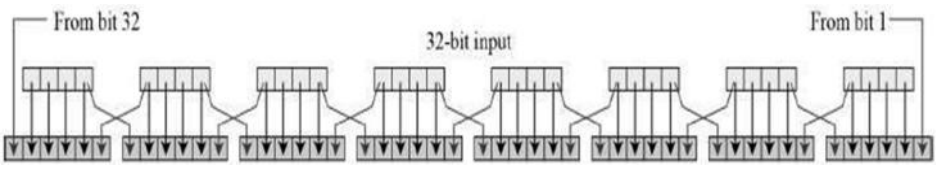


Round feature:

The heart of this cipher is the DES function, f. The DES function applies a 48-bit key to the rightmost 32 bits to produce 32-bit output.



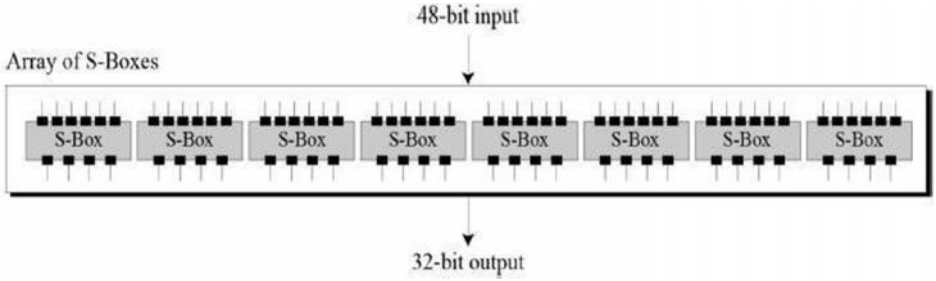
- Expansion permutation box: Since the correct input is 32 bits and the round key is 48 bits, we must first expand the correct input to 48 bits. The permutation logic is represented graphically in the following illustration:



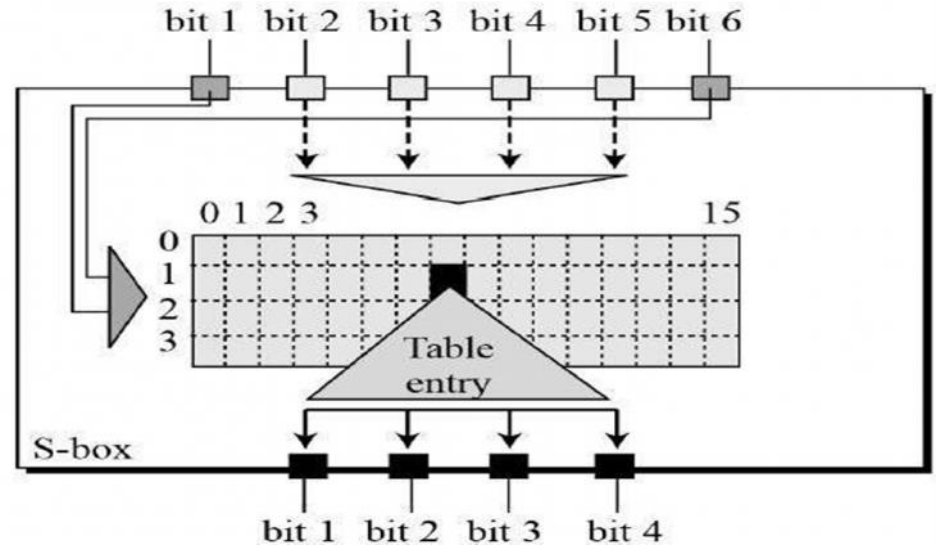
- The graphed permutation logic is generally described as a table in the DES specification illustrated as shown:

32	01	02	03	04	05
04	05	06	07	08	09
08	09	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	31	31	32	01

- XOR (bleach). - After the expansion permutation, DES performs the XOR operation on the expanded right section and round key. The round key is used only in this operation.
- Spare boxes. - The S-boxes do the actual mixing (confusion). DES uses 8 S boxes, each with a 6-bit input and a 4-bit output. Please refer to the illustration below:



- The S-box rule is illustrated below:

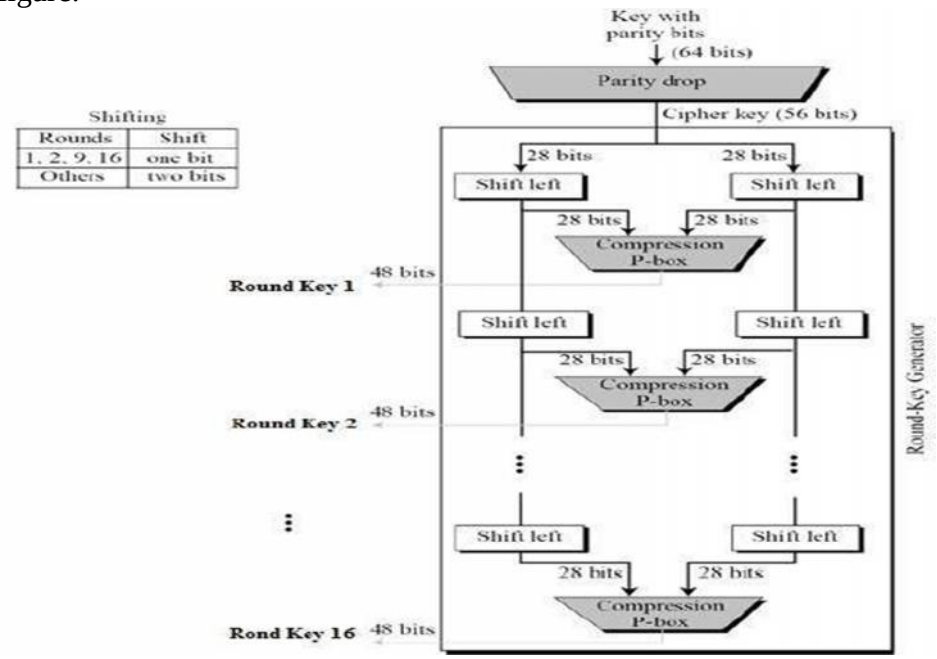


- There are a total of eight S-box tables. The output of the eight S-boxes is then combined into a 32-bit section.
- Forward permutation: The 32-bit output from the S-boxes is then subjected to forward permutation using the rule shown in the figure below:

16	07	20	21	29	12	28	17
01	15	23	26	05	18	31	10
02	08	24	14	32	27	03	09
19	13	30	06	22	11	04	25

Key generation:

The circular key generator creates sixteen 48-bit keys from a 56-bit encryption key. The key generation process is described in the following figure:



The logic for parity reduction, offset and compression P-box is provided in the DES description.

DES analysis:

DES satisfies the two desired properties of block cryptography. These two properties make the encryption very strong.

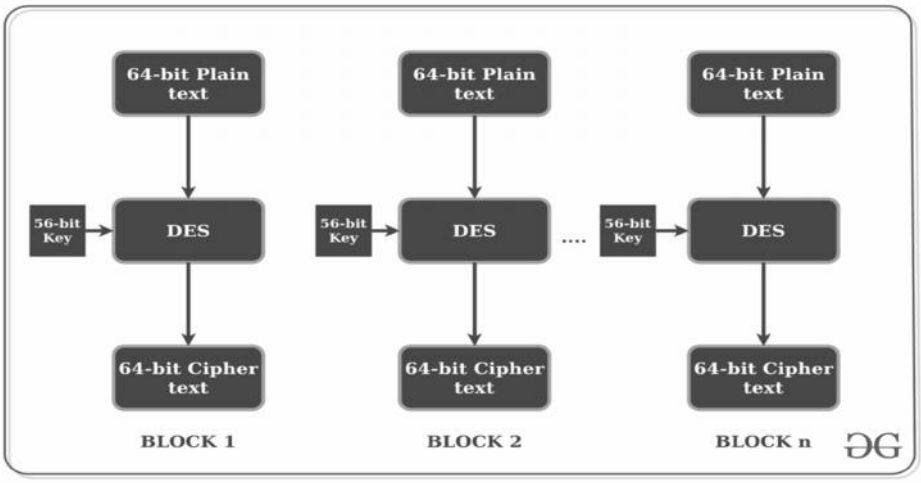
- Avalanche effect: A small change in the plaintext results in a large change in the ciphertext.
- Integrity: Each bit of ciphertext depends on many bits of plain text.

In recent years, cryptanalysis has found some weaknesses in DES when the selected keys are weak keys. These keys should be avoided.

DES proved to be a very well designed block cipher. There have been no significant cryptanalytic attacks on DES, aside from the exhaustive key search.

1.19 DES ENCRYPTION EXAMPLE

DES is a block cipher and encrypts data in blocks of 64 bits each, which means that 64 bits of plain text go as input to DES, which produces 64 bits of ciphertext. The same algorithm and key are used for encryption and decryption, with minor differences. The key length is 56 bits. The basic idea is shown in the figure.



We said that DES uses a 56-bit key. In reality, the initial key consists of 64 bits. However, even before the DES process begins, every eighth bit of the key is discarded to produce a 56-bit key. That is, bit positions 8, 16, 24, 32, 40, 48, 56 and 64 are discarded.

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

Figure - discording of every 8th bit of original key

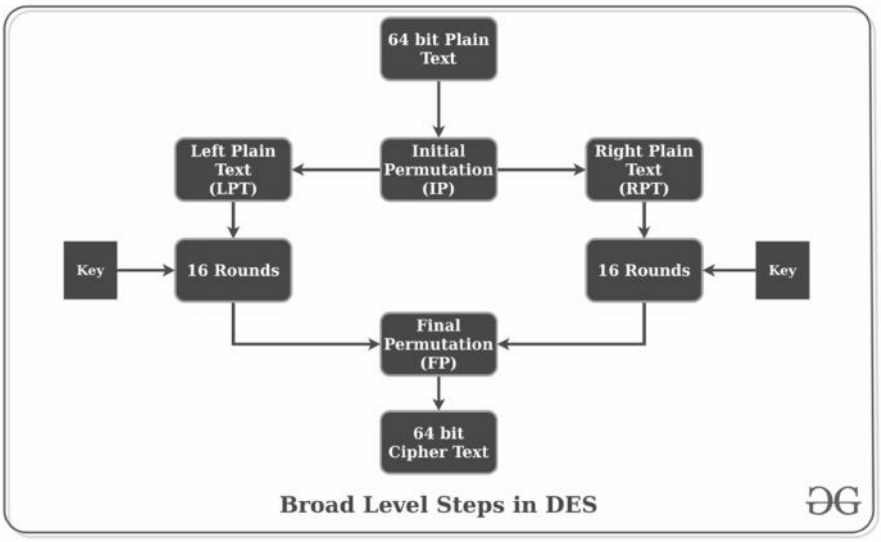
Therefore, discarding every eighth bit of the key produces a 56-bit key from the original 64-bit key.

DES is based on the two fundamental attributes of cryptography: substitution (also called confusion) and transposition (also called diffusion). DES is made up of 16 steps, each of which is called a round.

Each round performs the substitution and transposition steps. Let's now take a look at the general DES steps.

1. In the first step, the 64-bit plain text block is passed to an initial permutation (IP) function.
2. The initial permutation performed in plain text.

3. The initial permutation (IP) then produces two halves of the permuted block; Says Left Plain Text (LPT) and Right Plain Text (RPT).
4. Now each LPT and RPT goes through 16 encryption cycles.
5. Eventually, LPT and RPT are rejoined and a final permutation (FP) is performed on the combined block
6. The result of this process produces 64-bit ciphertext.



Initial Permutation (IP):

As we noted, the initial permutation (IP) occurs only once and occurs before the first round. Suggest how the PI transposition should proceed, as shown in the figure.

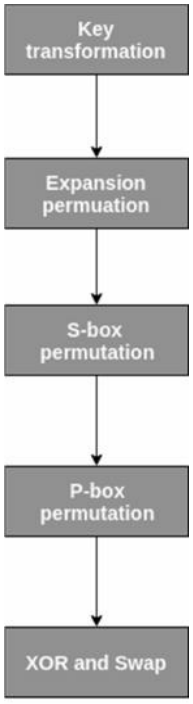
For example, it says the IP replaces the first bit of the original plaintext block with bit 58 of the original plaintext, the second bit with bit 50 of the original plaintext block, and so on.

This is nothing more than juggling the bit positions of the original text block. The same rule applies to all the other bit positions shown in the figure.

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

Figure - Initial permutation table

As we noted after the IP was done, the resulting 64-bit permuted text block is split into two halves. Each half block consists of 32 bits, and each of the 16 rounds, in turn, consists of the general level steps described in the figure.



Step 1: key transformation:

We noticed that the initial 64-bit key is transformed into a 56-bit key by discarding every eighth bit of the initial key. Therefore, a 56-bit key is available for each. From this 56-bit key, a different 48-bit subkey is generated during each round by a process called key transformation. To do this, the 56-bit key is divided into two halves, each of 28 bits. These halves move circularly to the left in one or two positions, depending on the round.

For example, if lap number 1, 2, 9 or 16 is changed only in position for the other laps, the circular change is made in two positions.

The number of key bits shifted per round is shown in the figure.

Round	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
#key bits shifted	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Figure - number of key bits shifted per round

After an appropriate shift, 48 of the 56 bits are selected. To select 48 bits out of 56, the table shows in the following figure. For example, after the move, bit number 14 moves to the first position, bit 17 moves to the second position, and so on. If we look closely at the table, we will realize that it only contains 48-bit positions. Bit number 18 is discarded (we won't find it in the table), like 7 others, to reduce a 56-bit key to a 48-bit key. Since the key transformation process involves both permutation and selection of a 48-bit subset of the original 56-bit key, it is called Compression Permutation.

14	17	11	24	1	5	3	28	15	6	21	10
23	19	12	4	26	8	16	7	27	20	13	2
41	52	31	37	47	55	30	40	51	45	33	48
44	49	39	56	34	53	46	42	50	36	29	32

Figure - compression permutation

Because of this compression permutation technique, a different subset of key bits are used in each round. This makes DES not easy to crack.

Step 2: permutation of expansion:

Remember that after the initial permutation, we had two 32-bit plain text areas called Left Plain Text (LPT) and Right Plain Text (RPT). During diffusion permutation, the RPT expands from 32 bits to 48 bits. The bits are permuted and are therefore called diffuse permutation. This happens when the 32-bit RPT is divided into 8 blocks and each block consists of 4 bits. Then each 4-bit block from the previous step is expanded to a corresponding 6-bit block, that is, per 4-bit block, another 2 bits are added.

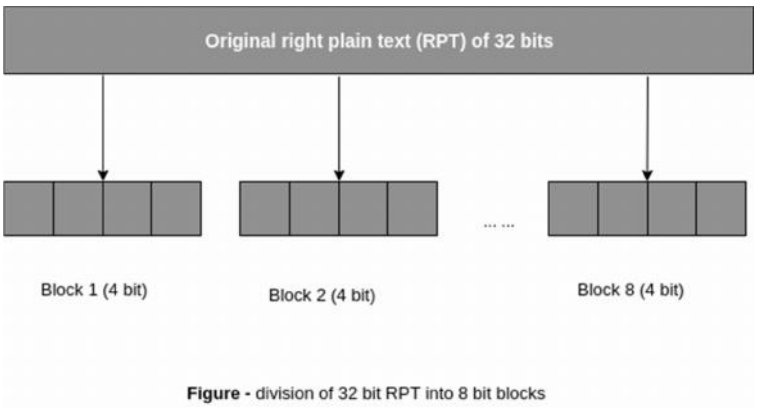


Figure - division of 32 bit RPT into 8 bit blocks

This process involves expanding and permuting the input bit while creating the output. The key transformation process compresses the key from 56 bits to 48 bits. Then the expansion permutation process expands the RPT from 32 bits to 48 bits. Now the 48-bit key is XOR with 48-bit RPT and the resulting output is given to the next step, which is the replacement of the S-Box.

1.20 OPERATING MODE OFF

Data encryption experts using DES can choose from five different operating modes.

- Electronic Code Book (ECB). Each 64-bit block is independently encrypted and decrypted

- Cipher Block Chaining (CBC). Each 64-bit block depends on the previous one and uses an initialization vector (IV)
- Cryptography Feedback (CFB). The ciphertext above becomes the input to the cipher algorithm, producing a pseudo-random output, which in turn is XORed with plaintext, building the following ciphertext unit
- Output feedback (OFB). Like CFB, except the cryptographic algorithm input is the DES output above
- Counter (CTR). Each block of plaintext is XORed with an encrypted counter. Then the counter is incremented for each subsequent block.

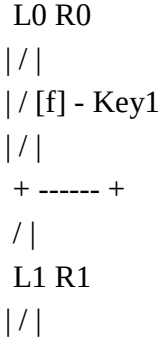
1.21 DIFFERENTIAL CRYPTANALYSIS

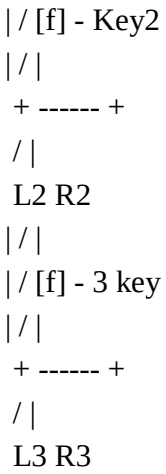
- Differential cryptanalysis preceded linear cryptanalysis and was initially designed in 1990 as an attack on DES. Differential cryptanalysis is similar to linear cryptanalysis; Differential cryptanalysis aims to map bitwise differences in inputs to differences in output in order to decode the action of the encryption algorithm. Again, the goal is to approximate the encryption algorithm by trying to find a maximum likelihood estimator of the true encryption action by altering the plaintext or (looking at a different plaintext) and analyzing the impact of the changes in the plaintext over the resulting ciphertext. Differential cryptanalysis is therefore a chosen plaintext attack.
- The description of differential cryptanalysis is analogous to that of linear cryptanalysis and is essentially the same as applying linear cryptanalysis to input differences rather than to input and output bits directly.

1.22 3-ROUND DES DIFFERENTIAL CRYPTANALYSIS

Description of 3 round DES 3:

DES is a Feistel network. Encryption is performed using F-Boxes which receive subkeys generated in the primary key. The following diagram omits the start and end permutations and the left-to-right swap on the output.





Principles of differential cryptanalysis:

There are several ideas intertwined in differential cryptanalysis.

- One is that following the xor difference of two inputs (called the differential) allows a deeper insight into the DES calculation, first canceling the effect of Li's xor sum on f (Ri, Keyi), and again canceling the xor sum of the key inside the F-Box.
- A second is that the round key is limited by knowledge of the differential input and differential output of the F box.
- Third (although we don't use it here), there is an imbalance in the frequency of the F-Box's differential outputs which allows for probabilistic tracking of the differentials when attacking more than three rounds.

3-round DES attack (mini-des):

The last stage is attached. Note that R2, the input of the last box F, is available on the output, like L3. If we knew L2, the output of the last F-Box would be known, as R3 + L2, and we could start forcing Key3 to open. We do not know about L2 as this would require knowledge of R1 and therefore knowledge of the output of the first F-Box. However, if we give two inputs (L0, R0) and (L0 ', R0') and also R0 = R0 ', then we know L2 + L2',

L2 + L2 '= R1 + R1' = (L0 + f (R0)) + (L0 '+ f (R0')) = L0 + L0 '

So we know the differential output of the F box from the third round, as well as its differential input.

R2 + R2 '= L3 + L3'.

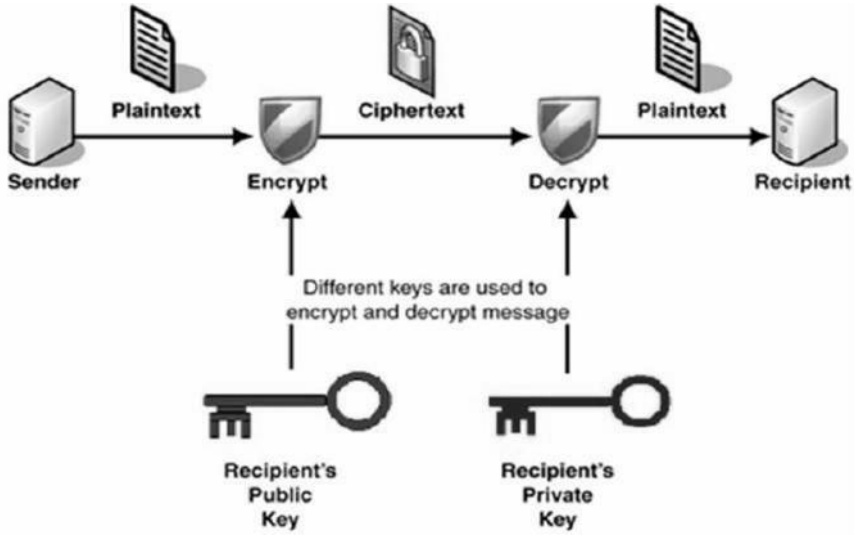
Given the differential input and output to an S-Box (when plotting the inputs and outputs to the F-Box via the E and P functions within the F-Box, again we use the xor jump property of the differentials), we get a List the possible input pairs: input pairs X and X 'such that X + X' is equal to

the differential input and f (X) + f (X ') is equal to the differential output. Then we compute the keys that transform X and X 'into R2 and R2'.

1.23 PUBLIC KEY CRYPTOGRAPHY

- Unlike symmetric key cryptography, we have not found a historical use of public key cryptography. It is a relatively new concept.
- Symmetric cryptography was suitable for organizations such as governments, the military, and large financial corporations involved in classified communication.
- With the spread of more insecure computer networks in recent decades, there has been a genuine need to use cryptocurrencies on a larger scale. The symmetric key was found to be impractical due to the challenges it faced in key management. This gave rise to public key cryptosystems.

The encryption and decryption process is shown in the figure below:



The most important properties of the public key cryptographic scheme are:

- Different keys are used for encryption and decryption. This is a property that sets this scheme differently from a symmetric encryption scheme.
- Each recipient has a unique decryption key, usually known as a private key.
- The recipient must publish an encryption key, called a public key.
- In this scheme, some guarantee of the authenticity of a public key is needed to avoid forgery by an adversary as a recipient. Typically, this type of cryptosystem involves a trusted third party certifying that a particular public key belongs only to a specific person or entity.

- The encryption algorithm is complex enough to prevent an attacker from inferring the plaintext from the ciphertext and (public) encryption key.
- Although the public and private keys are mathematically related, it is not possible to calculate the private key from the public key. In fact, the smart part of any public key cryptographic system is designing a relationship between two keys.

There are three types of public key cryptographic schemes. We discuss them in the following sections.

RSA cryptographic system:

This cryptosystem is one of the early systems. It remains the most used cryptosystem even today. The system was invented by three scholars Ron Rivest, Adi Shamir and Len Adleman and is therefore called the RSA cryptosystem.

We will analyze two aspects of the RSA cryptosystem, first, the generation of a key pair and, second, the encryption-decryption algorithms.

Generation of RSA key pairs:

Each person or party wishing to participate in the cryptographic communication must generate a key pair, i.e. public key and private key. The process followed in key generation is described below:

- Generate RSA form (n)
  - Pick two large prime numbers, eg q.
  - Compute n = p * q. For strong, indestructible encryption, let n be a large number, usually a minimum of 512 bits.
- Find the derived number (e)
  - The number e must be greater than 1 and less than (p - 1) (q - 1).
  - There should be no common factor for ey (p - 1) (q - 1) except 1.

In other words, two numbers and y (p - 1) (q - 1) are coprime.

- Form the public key
  - The pair of numbers (n, e) constitutes the RSA public key and is made public.
  - Interestingly, although n is part of the public key, the difficulty of factoring a large prime number ensures that an attacker cannot find the two primes (p & q) used to obtain n in finite time. This is the strength of RSA.
- Generate the private key
  - The private key d is computed from p, q and e. For n and e given, there is a unique number d.

- The number d is the inverse of e modulo (p - 1) (q - 1). This means that d is the smaller number of (p - 1) (q - 1) such that multiplied by e equals 1 modulo (p - 1) (q - 1).
- This relationship is mathematically written as follows:

  ed = 1 mod (p - 1) (q - 1)

The extended Euclidean algorithm takes p, q and e as input and gives d as output.

Example:

Below is an example of generating an RSA key pair. (For ease of understanding, the primes p & q taken here are small values. In practice, these values are very high).

- Let p = 7 and q = 13 be two prime numbers, so modulo n = pq = 7 x 13 = 91.
- Select e = 5, which is a valid choice since there is no number that is a common factor of 5 and (p - 1) (q - 1) = 6 × 12 = 72, except 1.
- The pair of numbers (n, e) = (91, 5) constitutes the public key and can be made available to anyone we wish to be able to send us encrypted messages.
- Enter p = 7, q = 13, and e = 5 in the extended Euclidean algorithm. The output will be d = 29.
- Verify that the calculated d is correct by calculating -

de = 29 × 5 = 145 = 1 mod 72

- Therefore, the public key is (91, 5) and the private key is (91, 29).

Encryption and decryption:

Once the key pair has been generated, the encryption and decryption process is relatively straightforward and computationally straightforward.

Interestingly, RSA does not operate directly on bit strings as is the case with symmetric key cryptography. Work in numbers module no. Therefore, it is necessary to represent the plaintext as a series of numbers less than n.

RSA encryption:

- Suppose the sender wants to send a text message to someone whose public key is (n, e).
- The sender then renders the plaintext as a series of numbers less than n.

- To encrypt the first plaintext P, which is a modulo number n. The encryption process is a simple mathematical step like:

  C = Pe form no

- In other words, the ciphertext C is equal to the plaintext P multiplied by itself and times and then reduced modulo n. This means that C is also a number less than n.
- Returning to our example of generating keys with plaintext P = 10, we get the ciphertext C -
  C = 105 mod 91

RSA decryption:

- The RSA decryption process is also very simple. Suppose the recipient of a public key pair (n, e) received a ciphertext C.
- The recipient raises C to the power of his private key d. The result modulo n will be the plain text P.

  Plain text = Cd mod n

- Returning to our numerical example, the ciphertext C = 82 would be decrypted at number 10 using the private key 29 -

  Plain text = 8229 mod 91 = 10

RSA analysis:

RSA security depends on the strengths of two independent functions. The RSA cryptosystem is the strong point of the most widespread public key cryptosystem of which it is based on the practical difficulty of factoring very large numbers.

- **Encryption function:** it is considered a one-way function to convert plain text to ciphertext and can only be canceled with the knowledge of the private key d.
- **Key Generation:** The difficulty of determining a private key from an RSA public key is equivalent to module n factoring. Therefore, an attacker cannot use knowledge of an RSA public key to determine an RSA private key unless they are able to factor n. It is also a one-way function, switching from p & q values to modulo n is easy but reversing is not possible.

If either of these two functions are found to be non-unidirectional, RSA will break. In fact, if you develop a technique to factor efficiently, RSA will no longer be safe.

The strength of RSA encryption is drastically reduced against attacks if the number p and q are not large primes and / or the chosen public key e is a small number.

ElGamal Cryptosystem:

Along with RSA, other public key cryptosystems are proposed. Many of them are based on different versions of the discrete logarithm problem.

The ElGamal cryptosystem, called Elliptic Curve Variant, is based on the discrete logarithm problem. It draws strength from the assumption that discrete logarithms cannot be found in a practical time frame for a given number, while the inverse operation of the power can be calculated efficiently.

Let's take a look at a simple version of ElGamal that works with modulo p numbers. In the case of the variants of the elliptic curve, it is based on quite different number systems.

Generation of the ElGamal key pair:

Each user of the ElGamal cryptosystem generates the key pair as follows:

- Choice of a great prize p. Generally a prime number with a length between 1024 and 2048 bits is chosen.
- Choice of a generator element g.
  - This number must be between 1 and p - 1, but it cannot be a number.
  - It is a generator of the multiplicative group of integers modulo p. This means that for every integer m coprimo ap, there exists an integer k such that gk = a mod n. For example, 3 is the generator of group 5 (Z5 = {1, 2, 3, 4}).

North	**3n**	**3n mod 5**
1	3	3
Two	9	4
3	27	Two
4	81	1

- Choice of private key. The private key x is any number greater than 1 and less than p - 1.
- Calculate part of the public key. The value y is calculated from the parameters p, g and from the private key x as follows:y = gx mod p
- Get an audiencekey. The public key of ElGamal consists of the three parameters (p, g, y), for example suppose that p = 17 and that g = 6 (it can be confirmed that 6 is a generator of the group Z17). The private key x can be any number greater than 1 and less than 71, so we choose x = 5. The y value is calculated as follows: y = 65 mod 17 = 7
- Therefore, the private key is 62 and the public key is (17, 6, 7).

Encryption and decryption:

Generating an ElGamal key pair is relatively simpler than the equivalent process for RSA. But encryption and decryption are a little more complex than RSA.

ElGamal encryption:

Suppose the sender wants to send plaintext to someone whose ElGamal public key is (p, g, y), then -

- The sender represents the plaintext as a series of modulo numbers p.
- To encrypt the first plaintext P, which is represented as a modulo number p. The encryption process for obtaining ciphertext C is as follows:
  - Randomly generate a number k;
  - Calculate two values C1 and C2, where -

  C1 = gk mod p

  C2 = (P * yk) mod p

- Send ciphertext C, which consists of two separate values (C1, C2), sent together.
- Referring to our ElGamal key generation example above, the plaintext P = 13 is encrypted as follows:
  - Randomly generate a number, let's say k = 10
  - Calculate the two values C1 and C2, where -

  C1 = 610 mod 17

  C2 = (13 * 710) modulo 17 = 9

- Send the ciphertext C = (C1, C2) = (15, 9).

ElGamal decryption

- To decrypt the ciphertext (C1, C2) using the private key x, follow the following two steps:
  - Find the modular inverse of (C1) x modulo p, which is (C1) -x, usually called the decryption factor.
  - Get the plain text using the following formula:
    C2 × (C1) -x mod p = plain text
- In our example, to decrypt the ciphertext C = (C1, C2) = (15, 9) using the private key x = 5, the decryption factor is 15-5 mod 17 = 9
- Extract plain text P = (9 × 9) mod 17 = 13.

ElGamal analysis:

In ElGamalsystem, each user has a private key x. and has three components of the public key: main module p, generator g and public Y = gx mod p. ElGamal's strength is based on the difficulty of discrete logarithm problems.

The size of the security key is generally> 1024 bits. Keys of 2048 bits in length are also used today. On the processing speed front, Elgamal is quite slow, mainly used for key authentication protocols. Due to the higher processing efficiency, ElGamal's elliptical curve variants are becoming more and more popular.

Elliptic Curve Encryption (ECC):

- Elliptic Curve Cryptography (ECC) is a term used to describe a set of cryptographic tools and protocols whose security is based on special versions of the discrete logarithm problem. Does not use form numbers p.
- ECC is based on sets of numbers associated with mathematical objects called elliptic curves. There are rules for adding and calculating multiples of these numbers, as well as for numbers modulo p.
- ECC includes variations of many cryptographic schemes that were initially designed for modular numbers such as ElGamal encryption and digital signature algorithm.
- The discrete logarithm problem is thought to be much more difficult when applied to points on an elliptical curve. This requires modifying the modulo numbers pa at points on an elliptical curve. An equivalent level of security can also be obtained with shorter keys if variants based on elliptic curves are used.

Shorter keys have two advantages:

- Ease of key management
- Efficient processing

These advantages make the variants of cryptographic schemes based on elliptic curves very attractive for applications where processing resources are limited.

RSA and ElGamal schemes: a comparison:

Let's briefly compare the RSA and ElGamal schemes in various aspects.

RSA	El Gamal
It is more efficient for encryption.	It is more efficient for decryption.
It is less efficient for decryption.	It is more efficient for decryption.
For a particular level of security,	Very short keys are required for

long keys are required in RSA.	the same level of security.
It is widely accepted and used.	It is new and not very popular in the market.

1.24 THE EUCLIDEAN ALGORITHM

The Euclidean algorithm for finding the GCD (A, B) is as follows:

- If A = 0, then GCD (A, B) = B, since GCD (0, B) = B, and we can stop.
- If B = 0, then GCD (A, B) = A, since GCD (A, 0) = A, and we can stop.
- Write A as a remainder quotient (A = B ·Q + R)
- Find MCD (B, R) using the Euclidean algorithm since MCD (A, B) = MCD (B, R)

Example:

Find the GCF of 270 and 192

- A = 270, B = 192
- At 0
- YES 0
- Use long division to find that 270/192 = 1 with a remainder of 78. We can write this as: 270 = 192 * 1 +78

Find GCD (192.78), since GCD (270.192) = GCD (192.78)

- A = 192, B = 78
- At 0
- YES 0
- Use long division to find 192/78 = 2 with a remainder of 36. We can write this as:
- 192 = 78 * 2 + 36

Find MCD (78.36), since MCD (192.78) = MCD (78.36)

- A = 78, B = 36
- At 0
- YES 0
- Use long division to find that 78/36 = 2 with a remainder of 6. We can write this as:
- 78 = 36 * 2 + 6

Find GCD (36.6), since GCD (78.36) = GCD (36.6)
A = 36, B = 6

- At 0
- YES 0
- Use long division to find 36/6 = 6 with a remainder of 0. We can write this as:
- 36 = 6 * 6 + 0

Find GCF (6.0), since GCF (36.6) = GCF (6.0)

- A = 6, B = 0
- At 0
- B = 0, GCD (6.0) = 6

So we showed:

MCD (270.192) = MCD (192.78) = MCD (78.36) = MCD (36.6) = MCD (6.0) = 6

MCD (270.192) = 6

1.25 CHINESE REMAINDER THEOREM

We are given two matrices num [0..k-1] and rem [0..k-1]. In num [0..k-1], each pair is coprime (LCD for each pair is 1). We must find a minimum positive number x such that:

x% num [0] = rem [0],

x% num [1] = rem [1],

.....

x% num [k-1] = rem [k-1]

Basically, we are given k numbers which are coprime pairs and we are given the remainders of these numbers when an unknown number x is divided by them. We need to find the smallest possible value of x that produces residual data.

Examples:

Input: num [] = {5, 7}, rem [] = {1, 3}

Departure: 31

Explanation:

31 is the smallest number such that:

(1) When we divide it by 5, we get the remainder 1.

(2) When we divide it by 7, we get the remainder 3.

Input: num [] = {3, 4, 5}, rem [] = {2, 3, 1}

Departure: 11

Explanation:

11 is the smallest number such that:

(1) When we divide it by 3, we get the remainder 2.

(2) When we divide it by 4, we get the remainder 3.

(3) When we divide it by 5, we get the remainder 1.

Moreover, the Chinese theorem states that there is always an x that satisfies the given congruences.

Let num [0], num [1],… num [k-1] positive integers coprime in pairs. Then, for any given sequence of integers rem [0], rem [1], ... rem [k-1], there exists an integer x which solves the following system of simultaneous congruences.

$$\begin{cases} x \equiv rem[0] & (\text{mod } num[0]) \\ \quad \dots & \\ x \equiv rem[k-1] & (\text{mod } num[k-1]) \end{cases}$$

Furthermore, all solutions x of this system are congruent modulo the product, prod = num[0] * num[1] * ... * nun[k-1]. Hence

$$x \equiv y \pmod{num[i]}, \quad 0 \leq i \leq k-1 \quad \Longleftrightarrow \quad x \equiv y \pmod{prod}.$$

The first part is clear that there is an x. The second part basically states that all solutions (including the least) produce the same residue when dividing the by-products of n [0], num [1], .. num [k-1]. In the example above, the product is 3 * 4 * 5 = 60. And 11 is a solution, other solutions are 71, 131, .. and so on. All these solutions produce the same remainder when divided by 60, i.e. they are of the form 11 + m * 60 where m> = 0.

A naive approach to finding x is to start with 1 and increase it one by one and see if dividing it with elements given in num [] yields the corresponding residuals in rem []. Once we find an x, we return it.

1.26 RSA ALGORITHM

The RSA algorithm is a public key cryptography technique and is considered the most secure form of cryptography. It was invented by Rivest, Shamir and Adleman in 1978 and therefore called the RSA algorithm.

Algorithm:

The RSA algorithm has the following characteristics:

- The RSA algorithm is a popular enhancement to a finite field on integers, including prime numbers.
- The integers used by this method are large enough to be difficult to solve.
- There are two sets of keys in this algorithm: private key and public key.

You will need to follow the steps below to work on the RSA algorithm:

Step 1: Create the RSA form:

The initial procedure starts with selecting two prime numbers, namely p and q, and then calculates their product N, as shown:
N = p * q

Here, let N be the large number specified.

Step 2: derivative number (e):

Treat the number e as a derived number that must be greater than 1 and less than (p-1) and (q-1). The main condition will be that there must be no common factor of (p-1) and (q-1) except 1

Step 3: public key:

The specified pair of numbers n and e constitutes the RSA public key and is made public.

Step 4: private key:

The private key d is calculated from the numbers p, q, and e. The mathematical relationship between the numbers is as follows:
ed = 1 mod (p-1) (q-1)

The above formula is the basic formula for the extended Euclidean algorithm, which accepts p and q as input parameters.

Cryptographic formula:

Consider a sender sending the plain text message to someone whose public key is (n, e). To encrypt the plain text message in the specified scenario, use the following syntax:
C = Pe form no

Decryption formula

The decryption process is very simple and includes analysis for computation in a systematic approach. Considering that receiver C has the private key d, the modulus of the result will be calculated as:

Plain text = Cd mod n

1.27 ATTACKS ON RSA

Below is the list of some possible attacks on the RSA algorithm:

1. Plain text attack:

Plain text attacks fall into three categories

- **Short message attack:** In this type of attack, the attacker is assumed to know some blocks of the plain text message. If an attacker is aware of blocks of plaintext, they could try to encrypt the blocks of plaintext using the information and try to convert it to ciphertext. To avoid a short message attack, we can use filler bits for encryption.
- **Attack of the cyclist:** In the cyclist's attack, the reverse process is performed. An attacker assumes that the ciphertext consists of some permutation operations. If the assumption of the attacker turns out to be true, you can try the reverse process to get the plaintext from the ciphertext.
- **Attack with invisible message:**In some rare cases, some of the ciphertext turns out to be the same as the plaintext, i.e. the original text. This means that plain text is not hidden. This type of attack is called a non-hidden message attack.

2. Cryptographic attack chosen:

In this type of attack, the attacker can find the plaintext from the ciphertext using the extended Euclidean algorithm.

3. Factoring attack:

In factoring attack, the attacker impersonates the owners of the keys and, with the help of the stolen cryptographic data, decrypts the confidential data, bypassing the security of the system. This attack occurs on an RSA cryptographic library used to generate the RSA key. In this way, attackers can have the private keys of various security tokens, smart cards, motherboard chipsets by having the public key of a target.

1.28 RABIN CRYPTOSYSTEM

Rabin cryptosystemis a public key cryptosystem invented by Michael Rabin. It uses asymmetric key cryptography to communicate between two parties and encrypt the message.

The security of the Rabin cryptosystem is linked to the difficulty of factoring. It has the advantage over the others that the problem on which it is based has proved to be as difficult as factoring whole numbers. It also has the disadvantage that each output of the Rabin function can be generated from any of the four possible inputs. If each output is ciphertext, additional decryption complexity is required to identify which of the four possible inputs was true plaintext.

Steps into the Rabin cryptosystem
Key generation
1. Generates two very large prime numbers, p and q, which satisfy the condition $p \neq q \rightarrow p \equiv q \equiv 3 \pmod 4$ For example: p = 139 and q = 191

2. Calculate the value of nn = pq

3. Publish n as a public key and save p and q as a private key

Encryption

1.Get public key #
2. Convert the message to ASCII value. Then convert it to binary and expand the binary value with itself and change the binary value back to decimal m.
3. Code with the formula: C = m2 mod n

4. Send C to the recipient.

decoded

1 Accept C from the sender.

2. Specify a and b with Euclidean GCD extended such that, ap + bq = 1

3. Calculate res using the following formula: r = C (p + 1) / 4 mod ps = C (q + 1) / 4 mod q

4. Now calculate X and Y using the following formula: X = (apr + bqs) mod pY = (apr - bqs) mod q

5. The four roots are, m1 = X, m2 = -X, m3 = Y, m4 = -Y Now convert them to binary and divide them all in half.

6. Determine how the left and right halves are equal. Keep that binary in the middle and convert it to m decimal. Get the ASCII character for the decimal value m. The resulting character provides the correct message sent by the sender.

1.28 QUADRATIC SIEVE

The Dixon method has its drawback, we have to look for numbers that are B-soft. Here we will try to create these numbers a.

Quadratic residual:

of x can be defined as the remainder that remains after squaring a number (0 <k <x) modulo x.

Example : Quadratic residual of x = 17, k will pass from 0 <k <17

$1^2 = 1$ module 17

$2^2 = 4$ module 17

$3^2 = 9$ mod 17

$4^2 = 16$ mod 17

$5^2 = 8$ mod 17

$6^2 = 2$ mod 17

$7^2 = 15$ mod 17

$8^2 = 13$ mod 17

$9^2 = 13$ mod 17

$10^2 = 15$ mod 17

Hence, the set of quadratic residuals is {1, 2, 4, 8, 9, 13, 15, 16}

The point to note is that we will only have to iterate x / 2, since after the quadratic values are all the same value in reverse order.

11 = 11 mod 17 can be written as 11 = -6 mod 17

Using the same fact here

$11^2 = (-6)^2 \bmod 17 = 2 \bmod 17$, which is equivalent to $6^2 \bmod 17$

Back to our problem: we choose a random ay and then we want $a^2 - N$ to be B-Smooth for a set of prime numbers.

$a^2 - N = 0 \pmod p$, this is a quadratic residual equation and we found for which primes p is N a quadratic residue? we know a and we know n, but we don't know p. This can be calculated using Euler's criteria.

N ^ (p-1) / 2 mod p = 1 then p has a factor in N; otherwise no.

So for n = 90283 we can calculate B = {2, 3, 7, 17, 23, 29, 37, 41, 53}

Note that p = 5 is not on the whole, we can quickly verify this fact

90283 ^ (5–1) / 2 = $90283^2 \bmod 5 = 4$ or $9028^{32} \bmod 5 = -1$ and therefore 5 is not in the factor set.

90283 ^ (53–1) / 2 mod 53 = 1 and therefore 53 is in the set

We have now created a

We are trying to find the first number of N such that it is the prime factor of the base B chosen above

$(X + N)^2 - N = 0 \pmod p$

The above equation can be solved using Shanks Tonelli's algorithm for odd prime. For p = 2, X can be 0 or 1 (remember your mod 2). Let me explain in some detail

$(X + N)^2 - N = 0 \pmod 2$

N is odd, so $(X + N)^2$ should also be odd [remember that odd -even is even and how much is 0 mod 2]

Now, because $(X + N)^2$ is odd, X + N is odd [remember that odd * odd is odd]

For X + N to be odd, if N is odd, then X = 0

An easy way to remember is

N = odd therefore X = 0

N = even then X = 1

If N is even, the above rule is reversed.

Example: 90283 since it is odd X = 0 that is $a = \sqrt{90283} = 301$

$b = 90283 - 301^2 = 318$ which has 2 as a prime factor.

So we start from 301 and all 2p, 3p (303, 305) will have 2 as prime factor.

For all other prime numbers, we can use Shanks Tonelli's algorithms to calculate the starting point of those prime numbers and then skip 2p, 3p.

Example: 90283 and we know from Euler's criterion that 3 is one of its prime factors

$(X + N)^2 - N = 0 \pmod 3$ => X will be 0 and 1

$301^2 - 90283 = 318$ which has 3 as a prime factor

$302^2 - 90283 = 921$ which also has 3 as a prime factor.

We see 2p i.e. 6 i.e. 301 + 6

$307^2 - 90283 = 3966$, this is also 3 as a prime factor

$308^2 - 90283 = 4581$, which also has 3 as a prime factor

*****

UNIT-II

2

SIGNATURE SCHEMES

Unit Structure

2.0 OBJECTIVES

In this chapter you will learn about:

- ElGamal Signature scheme
- Digital signature standard
- What is One-time signature?
- Undeniable and Fail-stop signature

2.1 INTRODUCTION

A "conventional" handwritten signature attached to a document is used to specify the person responsible for it. A signature is used in everyday situations such as writing a letter, withdrawing money from a bank, signing a contract, etc. A signature scheme is a method of signing a message stored in electronic form.

As such, a signed message can be transmitted over a computer network. In this chapter, we will study several signature schemes, but first we discuss some fundamental differences between conventional and digital signatures. First is the question of signing a document. With a conventional signature, a signature is part of the physical document being signed. However, a digital signature is not attached physically to the message that is signed, so the algorithm that is used must somehow "bind" the signature to the message.

Second is the question of verification. A conventional signature is verified by comparing it to other, authentic signatures. For example, if someone signs a credit card purchase (which is not so common nowadays, given the prevalence of chipand- pin technologies), the salesperson is supposed to compare the signature on the sales slip to the signature on the back of the credit card in order to verify the signature. Of course, this is not a very secure method as it is relatively easy to forge someone else's signature. Digital signatures, on the other hand, can be verified using a publicly known verification algorithm. Thus, "anyone" can verify a digital signature.

2.2 THE ELGAMAL SIGNATURE SCHEME

In this section, we present the ElGamal Signature Scheme, which was described in a 1985 paper. A modification of this scheme has been adopted as the Digital Signature Algorithm (or DSA) by the National Institute of Standards and Technology. The DSA also incorporates some ideas used in a scheme known as the Schnorr Signature Scheme. All of these schemes are designed specifically for the purpose of signatures, as opposed to the RSA Cryptosystem, which can be used both as a public-key cryptosystem and a signature scheme.

Let p be a prime such that the discrete log problem in $\mathbb{Z}_p$ is intractable and let $\sim \in \mathbb{Z}_p^*$ be a primitive element Let $P = \mathbb{Z}_p^*, A = \mathbb{Z}_p^* \times \mathbb{Z}_{p-1}$ and define

$$\mathrm{K} = \left\{ \left(p, \sim, a \right) : \equiv \sim^{a} (\mathrm{mod}\ p) \right\}.$$

The values p, $\sim$ and are the public key, and a is the private key.
For P = (p, $\sim$, a,) and for the (secret) random number $k = \mathbb{Z}_{p-1}^*$ define

$$sig_{\mathrm{K}} = (¥, ¢),$$

Where

$$¥ = \sim^{k} \bmod p$$

And

$$¢ = \left(x - a¥ \right)^{k-1} \bmod \left(p - 1 \right)$$

For $x, ¥, \in \mathbb{Z}_{p-1}$, $x, ¥, \in \mathbb{Z}_p^*$ *and* $¢ \in \mathbb{Z}_{p-1}$, define $x, ¥, \in \mathbb{Z}_{p-1}$, define

$ver_k\left(x, (¥, ¢)\right) = true \Leftrightarrow \quad ^{¥}¥^{¢} \equiv \sim^{x} (\mathrm{mod}\ p)$

ElGamal Signature Scheme The ElGamal Signature Scheme is randomized (recall that the ElGamal Publickey Cryptosystem is also randomized). This means that there are many valid signatures for any given message, and the verification algorithm must be able to accept any of these valid signatures as authentic. The description of the ElGamal Signature Scheme is given as Cryptosystem. We begin with a couple of preliminary observations. An ElGamal signature consists of two

components, which are denoted *g* and *d*. The first component, *g*, is obtained by raising *a* to a random power modulo p; it does not depend on the message (namely, x) that is being signed. The second component, *d*, depends on the message x as well as the private key a. Verifying the signature is accomplished by checking that a certain congruence holds modulo p; this congruence does not involve the private key, of course. We now show that, if the signature was constructed correctly, then the verification will succeed. This follows easily from the following congruencies:

$$\equiv \sim^{a¥} \sim^{k¢} (\text{mod } p)$$

$$\equiv \sim^{x} (\text{mod } p)$$

Where we use the fact that

$$a¥ + k¢ \equiv x(\text{mod } p-1)$$

Actually, it is probably less mysterious to begin with the verification equation, and then derive the corresponding signing function. Suppose we start with the congruence

$$\sim^{x} \equiv \quad ^{¥}¥^{¢} (\text{mod } p)$$

Then we make the substitutions

$$¥ \equiv \sim^{k} (\text{mod } p)$$

And

$$\equiv \sim^{a} (\text{mod } p)$$

but we do not substitute for g in the exponent of (8.1).We obtain the following:

$$\sim^{x} \equiv \sim^{a¥+k¢} (\text{mod } p)$$

Now, a is a primitive element modulo p; so this congruence is true if and only if the exponents are congruent modulo p - 1, i.e., if and only if

$$x \equiv \sim¥ + k¢ (\text{mod } p-1)$$

Given x, a, *g*, and k, this congruence can be solved for *d*, yielding the formula used in the signing function of Cryptosystem.

Alice computes a signature using both the private key, a, and the secret random number, k (which is used to sign one message, x). The verification can be accomplished using only public information. Let's do a small example to illustrate the arithmetic.

Example 2.1

Suppose we take p=467, $\sim = 2$, a=127; then

$$= \sim^{a} \text{mod } p$$

$$= 2^{127} \text{ mod } 467$$

= 132

Suppose Alice wants to sign the message x = 100 and she chooses the random value k=213(note that gcd(213,466)=1 and 213^{-1}mod 466=431). Then

$$¥ = 2^{213} \bmod 467 = 29$$

And

$$¢ = (100 - 127 \times 29)\,431 \bmod 466 = 51$$

Any one can verify signature (29,51) by checking that

$$132^{29} 29^{51} \equiv 189 (\bmod 467)$$

And

$$2^{100} \equiv 189 (\bmod 467)$$

Hence, the signature is valid

2.3 THE DIGITAL SIGNATURE STANDARD

The **Digital Signature Standard** (or **DSS**) is a modification of the **ElGamal Signature Scheme**. It was published in the Federal Register on May 19, 1994 and adopted as a standard on December 1, 1994 (however, it was first proposed in August, 1991). First, we want to motivate the changes that are made to **ElGamal**, and then we will describe how they are accomplished.

In many situations, a message might be encrypted and decrypted only once, so it suffices to use any cryptosystem which is known to be secure at the time the message is encrypted. On the other hand, a signed message could function as a legal document such as a contract or will, so it is very likely that it would be necessary to verify a signature many years after the message is signed. So it is important to take even more precautions regarding the security of a signature scheme as opposed to a cryptosystem.

Since the **ElGamal Scheme** is no more secure than the **Discrete Logarithm** problem, this necessitates the use of a large modulus *p*. Certainly *p* should have at least 512 bits, and many people would argue that the length of *p* should be 1024 bits in order to provide security into the foreseeable future.

However, even a 512 bit modulus leads to a signature having 1024 bits. For potential applications, many of which involve the use of smart cards, a shorter signature is desirable. **DSS** modifies the **ElGamal Scheme** in an ingenious way so that a 160-bit message is signed using a 320-bit signature, but the computations are done using a 512-bit modulus *p*. The way that this done is to work in a subgroup of Z size 2^1 . The assumed security of the scheme is based on the belief that finding discrete algorithms in this specified subgroup of Z is secure.

The first change we make is to change the "-" to a "+" in the definition of δ, so

$$¢ = (x + a¥)k^{-1} \bmod (p-1)$$

This changes the verification condition to the following:

$$\sim^{x\ \ ¥} \equiv ¥^{¢} \bmod (p)$$

If gcd(x + aγ, p - 1) = 1, then δ^{-1} mod (p - 1) exists, and we can modify condition (6.1), producing the following:

$$\sim^{x¢-1\ \ ¥¢-1} \equiv ¥ \bmod (p)$$

Now here is the major innovation in the **DSS**. We suppose that *q* is a 160-bit prime such that *q* | (*p* - 1), and α is a *q*th root of 1 modulo *p*. (It is easy to construct such an a: Let α_0 be a primitive element of Z_p , and define $\sim = \sim_0^{(p-1)q} \bmod (p)$. Then β and γ will also be *q*th roots of 1. Hence, any exponents of α, β and γ can be reduced modulo *q* without affecting verification condition (6.2). The tricky point is that g appears as an exponent on the left side of (6.2), and again — but not as an exponent — on the right side of (6.2). So if g is reduced modulo *q*, then we must also reduce the entire left side of (6.2) modulo *q* in order to perform the verification. Observe that (6.1) will not work if the extra reductions modulo *q* are done. The complete description of the **DSS**.

Notice that is necessary that $¢ \not\equiv 0 \pmod q$ since the value δ^{-1} mod *q* is needed to verify the signature (this is analogous to the requirement that gcd(δ, *p*-1) = 1 when we modified (6.1) to obtain (6.2)). If Bob computes a value d º 0 (mod *q*) in the signing algorithm, he should reject it and construct a new signature with a new random *k*. We should point out that this is not likely to cause a problem in practice: The probability that $\delta \equiv 0 \pmod q$ is likely to be on the order of 2^{-1} , so for all intents and purposes it will almost never happen. Here is a small example to illustrate

Example 2.2:

Suppose we take *q* = 101 and *p* = 78*q* +1=7879.3 is a primitive element in $\mathbb{Z}_{7879}$, so we can take

$$\sim = 3^{78} \bmod 7879 = 170$$

Suppose a=75; then

$$= \sim^{a} \bmod 7879 = 4567$$

Now, Suppose bob wants to sign the message x=22 and he chooses the random value k=50, so

$$k^{-1} \bmod 101 = 99$$

Then

$$¥ = \left(170^{50} \bmod 7879\right) \bmod 101$$
$$= 2518 \bmod 101$$
$$= 94$$

And

$$¢ = (22 + 75 \times 94)99 \bmod 101$$
$$= 97$$

The signature (94, 97) on the message 22 is verified by the following computations

$$¢^{-1} = 97^{-1} \bmod 101 = 25$$
$$e_1 = 22 \times 25 \bmod 101 = 45$$
$$e_2 = 94 \times 25 \bmod 101 = 27$$
$$\left(170^{45} 4567^{27} \bmod 7879\right) \bmod 101 = 2518 \bmod 101 = 94.$$

When the **DSS** was proposed in 1991, there were several criticisms put forward. One complaint was that the selection process by NIST was not public. The standard was developed by the National Security Agency (NSA) without the input of U. S. industry. Regardless of the merits of the resulting scheme, many people resented the "closed-door" approach.

Of the technical criticisms put forward, the most serious was that the size of the modulus *p* was fixed at 512 bits. Many people would prefer that the modulus size not be fixed, so that larger modulus sizes could be used if desired. In response to these comments, NIST altered the description of the standard so that a variety of modulus sizes are allowed, namely, any modulus size divisible by 64, in the range from 512 to 1024 bits.

2.4 ONE-TIME SIGNATURES

In this section, we describe a conceptually simple way to construct a one-time signature scheme from any one-way function. The term "one-time" means that only one message can be signed. (The signature can be verified an arbitrary number of times, of course.) The description of the scheme, known as the **Lamport Signature Scheme.**

Informally, this is how the system works. A message to be signed is a binary *k*-tuple. Each bit is signed individually: the value *zi*,*j* corresponds to the *i*th bit of the message having the value *j* (*j* = 0, 1). Each *zi*,*j* is the image of *yi*,*j* under the one-way function *f*. The *i*th bit of the message is signed using the preimage *yi*,*j* of the *zi*,*j* corresponding to the *i*th bit of the message. The verification consists simply of checking that each element in the signature is the preimage of the appropriate public key element. We illustrate the scheme by considering one possible implementation using the exponentiation function $f(x) = \alpha^x \bmod p$, where α is a primitive element modulo *p*.

Example 2.3

7879 is prime and 3 is primitive element in $\mathbb{Z}_{7879}$ Define

$$f(x) = 3^x \bmod 7879$$

Suppose Bob wishes to sign a message of three bits, and he chooses the six (secret) random numbers

$$y_{1,0} = 5831$$
$$y_{1,1} = 735$$
$$y_{2,0} = 803$$
$$y_{2,1} = 2467$$
$$y_{3,0} = 4285$$
$$y_{3,1} = 6449$$

Then he computes the images of the y 's under the function *f*

$$z_{1,0} = 2009$$
$$Z_{1,1} = 3810$$
$$Z_{2,0} = 4672$$
$$Z_{2,1} = 4721$$
$$Z_{3,0} = 268$$
$$y_{3,1} = 5731$$

These *z*'s are published. Now, suppose Bob wants to sign the message

$$X = (1,1,0)$$

The signature for *x* is

$$y_{1,1}, y_{2,1}, y_{3,0} = (735, 2467, 4285)$$

To verify this signature, it suffices to compute the following:

$$3^{735} \bmod 7879 = 3810$$
$$3^{2467} \bmod 7879 = 4721$$
$$3^{4285} \bmod 7879 = 268$$

Hence, the signature is valid.

Oscar cannot forge a signature because he is unable to invert the one-way function *f* to obtain the secret *y*'s. However, the signature scheme can be used to sign only one message. For, given signatures for two different messages, it is (usually) an easy matter for Oscar to construct signatures for further messages (different from the first two).

For example, suppose the messages (0, 1, 1) and (1, 0, 1) are both signed using the same scheme. The message (0, 1, 1) would have as its signature the triple ($y1$,0, $y2$,1, $y3$,1), and the message (1, 0, 1) would be signed with ($y1$,1, $y2$,0, $y3$,1). Given these two signatures, Oscar can

manufacture signatures for the messages (1, 1, 1) (namely, ($y1$,1, $y2$,1, $y3$,1)) and (0, 0, 1) (namely, ($y1$,0, $y2$,0, $y3$,1)). Even though this scheme is quite elegant, it is not of great practical use due to the size of the signatures it produces.

For example, if we use the modular exponentiation function, as in the example above, then a secure implementation would require that p be at least 512 bits in length. This means that each bit of the message is signed using 512 bits. Consequently, the signature is 512 times as long as the message. We now look at a modification due to Bos and Chaum that allows the signatures to be made somewhat shorter, with no loss of security. In the **Lamport Scheme**, the reason that Oscar cannot forge a signature on a (second) message, given a signature on one message, is that the y's corresponding to one message are never a subset of the y's corresponding to another (distinct) message.

Suppose we have a set $\boldsymbol{B}$ of subsets of a set B such that $B_2 \subseteq B$ only if $B1 = B2$, for all $B1$, $B_2 \subseteq B$ Then $\boldsymbol{B}$ is said to satisfy the *Sperner property*. Given a set B of even cardinality $2n$, it is known that the maximum size of a set $\boldsymbol{B}$ of subsets of B having the Sperner property is $\binom{2n}{n}$. This can easily be obtained by taking all the n-subsets of B: clearly no n-subset is contained in another n-subset.

Now suppose we want to sign a k-bit message, as before, and we choose n large enough so that $2^k \leq \binom{2n}{n}$

2.5 UNDENIABLE SIGNATURES

Undeniable signatures were introduced by Chaum and van Antwerpen in 1989. They have several novel features. Primary among these is that a signature cannot be verified without the cooperation of the signer, Bob. This protects Bob against the possibility that documents signed by him are duplicated and distributed electronically without his approval. The verification will be accomplished by means of a *challenge-and-response protocol*.

But if Bob's cooperation is required to verify a signature, what is to prevent Bob from disavowing a signature he made at an earlier time? Bob might claim that a valid signature is a forgery, and either refuse to verify it, or carry out the protocol in such a way that the signature will not be verified. To prevent this from happening, an undeniable signature scheme incorporates a *disavowal protocol* by which Bob can prove that a signature is a forgery.

Thus, Bob will be able to prove in court that a given forged signature is in fact a forgery. (If he refuses to take part in the disavowal protocol, this would be regarded as evidence that the signature is, in fact, genuine.) Thus, an undeniable signature scheme consists of three components: a signing algorithm, a verification protocol, and a disavowal protocol. First, we present the signing algorithm and verification protocol of the **Chaum-van Antwerpen Undeniable Signature Scheme**

Let p=2q+1 be a prime such that q is prime and the discrete log problem in $\mathbb{Z}_p^*$ is intractable, Let $\sim \in \mathbb{Z}_p^*$ be an element of order q. Let $1 \le a \le q-1$ and define $= \sim^a \bmod p$. Let G denote the multiplicative subgroup of $\mathbb{Z}_p^*$ of order q (G consists of quadratic residues modulo p). Let $\mathbb{Z}_p^*$ P = A = G and define

$$\mathrm{K} = \left\{ \left(p, \sim, a, \quad \right) : \quad \equiv \sim^a \left(\bmod p \right) \right\}$$

The values p, $and \sim$ are the public key, and a is the private key.

For $\mathrm{K} = \left(p, \sim, a, \quad \right)$ and $x \in G$, define

$$y = sig_k \left(x \right) = x^a \left(p, \sim, a, \quad \right) \bmod p.$$

For $x, y \in G,$ verification is done be executing the following protocol:

1) Bob chooses $e_1 e_2$ at random, $e_1 e_2 \in \mathbb{Z}_q$

2) Bob computes $c = y^{e_1, \quad e_2} \bmod$ p and sents it to Allice

3) Alice computes $d = c^{a^{-1} \bmod q \quad e_2} \bmod$ p and sends it to Bob

4) Bob accepts y as a valid signature if and only if

$$d \equiv x^{e_1} \sim^{e_2} \left(\bmod p \right).$$

Chaum-van Antwerpen Signature Scheme:

We should explain the roles of p and q in this scheme. The scheme lives in ; Z_p however, we need to be able to do computations in a multiplicative subgroup G of Z_p of prime order. In particular, we need to be able to compute inverses modulo |G|, which is why |G| should be prime. It is convenient to take $p = 2q + 1$ where q is prime. In this way, the subgroup G is as large as possible, which is desirable since messages and signatures are both elements of G.

We first prove that Alice will accept a valid signature. In the following computations, all exponents are to be reduced modulo q. First, observe that

$$d \equiv c^{a^{-1}} \left(\bmod p \right)$$
$$\equiv y^{e_1 a^{-1} \quad e 2 a^{-1}} \left(\bmod p \right)$$

Since

$$\equiv \sim^{a} \pmod{p}$$

We have that

$$^{a^{-1}} \equiv \sim \pmod{p}$$

Similarly

$$y = x^{a} \pmod{p}$$

Implies that

$$y^{a^{-1}} \equiv x \pmod{p}$$

Hence

$$d \equiv x^{e_1} \sim^{e_2} \pmod{p}$$

As desired

Here is a small example.

Example 2.4:

Suppose we take p = 467. Since 2 is a primitive element, 2^2 = 4 is a generator of G, the quadratic residues modulo 467. So we can take α = 4. Suppose a = 101; then

$$\equiv \sim^{a} \bmod 467 = 449$$

Bob will sign the message x = 119 with the signature

$$y \equiv 119^{101} \bmod 467 = 129$$

Now, suppose Alice wants to verify the signature y. Suppose she chooses the random values e_1= 38, e_2= 397. She will compute c = 13, whereupon Bob will respond with d = 9. Alice checks the response by verifying that

$$119^{38} 4^{397} \equiv 9 \pmod{467}$$

Hence, Alice accepts the signature as valid.

We next prove that Bob cannot fool Alice into accepting a fradulent signature as valid, except with a very small probability. This result does not depend on any computational assumptions, i.e., the security is unconditional.

2.6 FAIL-STOP SIGNATURES

A fail-stop signature scheme provides enhanced security against the possibility that a very powerful adversary might be able to forge a signature. In the event that Oscar is able to forge Bob's signature on a

message, Bob will (with high probability) subsequently be able to prove that Oscar's signature is a forgery.

In this section, we describe a fail-stop signature scheme constructed by van Heyst and Pedersen in 1992. This is a one-time scheme (only one message can be signed with a given key). The system consists of signing and verification algorithms, as well as a "proof of forgery" algorithm. The description of the signing and verification algorithms of the **van Heyst and Pedersen Fail-stop Signature Scheme**.

Let p=2q+1 be a prime such that q isa prime and the discrete log problem in $\mathbb{Z}_p$ is intractable. Let $\sim \in \mathbb{Z}_p^*$ be an element of order q. Let $1 \le a_0 \le q-1$ and define $= \sim^{a0} \bmod p$. The values p,q, ~, ,and a_0 are chosen by a central (trusted) authority. *p,q,~, and* ,are public and will be regarded as fixed. The value of a_0 is kept secret from everyone (even Alice).

Let $P = \mathbb{Z}_q$ and $A = \mathbb{Z}_q \times \mathbb{Z}_q$ A key has the form

$$k = \left(¥_{1,} ¥_{2,} a_{1,} a_{2,} b_{1,} b_{2,} \right)$$

Where $a_1, a_2, b_1, b_2, \in \mathbb{Z}_q$

$$¥_{1,} = \sim^{a_1 \quad a_2} \bmod p \text{ and } ¥_{2,} = \sim^{b_1 \quad b_2} \bmod p$$

$\left(¥_1, ¥_{2,} \right)$ is a public key and $\left(a_1, a_2, b_1, b_2 \right)$ is a private key

For $k = \left(¥_{1,} ¥_{2,} a_{1,} a_{2,} b_{1,} b_{2,} \right)$ and $x \in \mathbb{Z}_q$ define

$$sig_k \left(x \right) = \left(y_{1,} y_2 \right),$$

Where

$$y_{1,} = a_1 + xb_1 \bmod q \text{ and } y_{2,} = a_2 + xb_2 \bmod q$$

For $y = \left(y_{1,} y_{2,} \right) \in \mathbb{Z}_q \times \mathbb{Z}_q$, We have,

$verk\left(x, y \right) = true \Leftrightarrow ¥_1 ¥_2^{\ x} \equiv \sim^{y_1 \quad y_2} \left(\bmod p \right)$

It is straightforward to see that a signature produced by Bob will satisfy the verification condition, so let's turn to the security aspects of this scheme and how the fail-stop property works. First we establish some important facts relating to the keys of the scheme. We begin with a definition. Two keys (γ1, γ2, *a*1, *a*2, *b*1, *b*2) and are said to be *equivalent* if and . It is easy to see that there are exactly q^2 keys in any equivalence class.

We establish several lemmas. LEMMA 6.4

Suppose k and k` are equivalent key and suppose that $ver_k \left( x, y \right) = true$ then $ver_{k`} \left(x, y \right) = true$

Proof suppose $k \left(¥_1, ¥_2, a_1, a_2, b_1, b_2 \right)$ and $k` \left(¥_1, ¥_2, a'_1, a'_2, b'_1, b'_2 \right)$ where

$$`¥_1 = \sim^{a_1 \quad a_2} \bmod p = \sim^{a'_1 \quad a_{\cdot 2}} \bmod p$$

And

$$`¥_2 = \sim^{b_1 \quad b_2} \bmod p = \sim^{b'_1 \quad b'_2} \bmod p$$

Suppose x is signed using k producing the signature y= (y_1,y_2) where

$$y_1 = \sim_1 + xb_1 \bmod q,$$
$$y_2 = \sim_2 + xb_2 \bmod q,$$

Now suppose we verify y using k`:

$$\sim^{y_1` \quad y_2} \equiv \sim^{a'_1 + xb'_1 \quad a'_2 + xb'_2} \left(\bmod p\right)$$
$$\equiv \sim^{a'_1 \quad a'_2} \left(\sim^{b'_1 \quad b'_2}\right)^x \bmod p$$
$$\equiv ¥_1 ¥_2{}^x \left(\bmod p\right).$$

Thus, *y* will also be verified using *K′*.

LEMMA 6.5

Suppose k is a key and y= sig_k (x). Then there are exactly q keys K' equivalent to K such that y=sig_k(x).

Proof suppose $¥_1, ¥_2$ are the public components of k. We want to determine the number of 4-tuples (a_1, a_2, b_1, b_2) such that the following congruences are satisfied

$$¥_1 \equiv \sim^{a_1 \quad a_2} \left(\bmod p\right).$$
$$¥_2 \equiv \sim^{b_1 \quad b_2} \left(\bmod p\right).$$
$$y_1 \equiv a_1 + xb_1 \left(\bmod q\right).$$
$$y_2 \equiv a_2 + xb_2 \left(\bmod q\right).$$

Since ~ generates G, there exist unique exponents $c_1, c_2, a_0 \in \mathbb{Z}_q$ such that

$$¥_1 \equiv \sim^{c_1} \left(\bmod p\right),$$
$$¥_1 \equiv \sim^{c_2} \left(\bmod p\right),$$

And

$$\equiv \sim^{a_0} \left(\bmod p\right),$$

Hence, it is necessary and sufficient that the following system of congruences be satisfied

$$c_1 \equiv a_1 + a_0 a_2 \left(\bmod q\right).$$
$$c_2 \equiv b_1 + a_0 b_2 \left(\bmod q\right).$$
$$y_1 \equiv a_1 + xb_1 \left(\bmod q\right).$$
$$y_2 \equiv a_2 + xb_2 \left(\bmod q\right).$$

This system can in turn be written as a matrix equation in $\mathbb{Z}_q$ as follows

$$\begin{pmatrix} 1 & a_0 & 0 & 0 \\ 0 & 0 & 1 & a_0 \\ 1 & 0 & x & 0 \\ 0 & 1 & 0 & x \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ b_1 \\ b_2 \end{pmatrix} = \begin{pmatrix} c_1 \\ c_2 \\ y_1 \\ y_2 \end{pmatrix}$$

Now the coefficient matrix of this system can be seen to have rank 1 three. Clearly the rank is three since rows 1,2 and 4 are linearly independent over $\mathbb{Z}_q$. And the rank is at most three since

1. The rank of a matrix is maximum number of linearly independent rows it contains

$r_1 + xr_2 - r_3 - a_0 r_4 = (0,0,0,0)$,

where r_i denotes ith row of the matrix. Now, this system of equations has at least one solution, obtained by using the key K. Since the rank of the coefficient matrix is three, it follows that the dimension of the solution space is 4 - 3 = 1, and there are exactly q solutions

2.7 LET US SUM UP

Thus, we have studied the basic concepts about the standard of signature like Digital signature. Here some basics of ElGamal key cryptography scheme and some fail-stop, undeniable signature has been explained.

2.8 UNIT END EXERCISES

1. Explain the digital signature standard.
2. Write a short note on Fail-Stop Signature.
3. Write a short note on ElGamal key cryptography
4. What do you mean by Undeniable signature?

2.9 LIST OF REFERENCES

- **Cryptography: Theory and Practice,** *Douglas Stinson*, CRC Press, CRC Press LLC
- **Cryptography and Network Security Principles and Practices**, Fourth Edition, William Stallings, PHI(Pearson),

*****

UNIT-II

3

HASH FUNCTIONS

Unit Structure

3.0 OBJECTIVES

In this chapter you will learn about:

- How hash function used in cryptography?
- Some collision-free hash function
- Birthday attack
- Discrete log and extending has function
- MD4 hash function
- Time-stamping

3.1 INTRODUCTION

Cryptographic hash functions map input strings of arbitrary length to short fixed length output strings. They were introduced in cryptology in the 1976 seminal paper of Diffie and Hellman on public-key cryptography. Hash functions can be used in a broad range of applications: to compute a short unique identifier of a string (e.g. for a digital signature), as one-way function to hide a string (e.g. for password

protection), to commit to a string in a protocol, for key derivation and for entropy extraction.

Until the late 1980s, there were few hash function designs and most proposals were broken very quickly after their introduction. The first theoretical result is the construction of a collision-resistance hash function based on a collision-resistant compression function, proven independently by Damgard and Merkle in 1989. Around the same time, the first cryptographic algorithms were proposed that are intended to be fast in software; the hash functions MD4 and MD5 fall in this category

Cryptographic hash functions have grown to be some of the most widely-used objects from cryptography. Hash functions are one-way: we cannot reverse a hash value to find the original content. (Irreversible) If we pass the same content through the same hash function then it should produce the same output/same hash value.

3.2 SIGNATURES AND HASH FUNCTIONS

When using the **DSS**, a 160-bit message is signed with a 320-bit signature. In general, we will want to sign much longer messages. A legal document, for example, might be many megabytes in size.

A naive attempt to solve this problem would be to break a long message into 160-bit chunks, and then to sign each chunk independently. This is analogous to encrypting a long string of plaintext by encrypting each plaintext character independently using the same key (e.g., ECB mode in the **DES**).

But there are several problems with this approach in creating digital signatures. First of all, for a long message, we will end up with an enormous signature (twice as long as the original message in the case of the **DSS**). Another disadvantage is that most "secure" signature schemes are slow since they typically use complicated arithmetic operations such as modular exponentiation. But an even more serious problem with this approach is that the various chunks of a signed message could be rearranged, or some of them removed, and the resulting message would still be verified. We need to protect the integrity of the entire message, and this cannot be accomplished by independently signing little pieces of it.

The solution to all of these problems is to use a very fast public *cryptographic hash function*, which will take a message of arbitrary length and produce a *message digest* of a specified size (160 bits if the **DSS** is to be used). The message digest will then be signed. For the **DSS**, the use of a hash function h is depicted diagrammatically.

When Bob wants to sign a message x, he first constructs the message digest $z = h(x)$, and then computes the signature $y = sig_K(z)$. He

transmits the ordered pair (x, y) over the channel. Now the verification can be performed (by anyone) by first reconstructing the message digest $z = h(x)$ using the public hash function h, and then checking that $ver_K(z, y) =$ true.

message	x	$x \in \{0,1\}^*$
	$\downarrow$	
message digest	$z = h(x)$	$z \in Z$
	$\downarrow$	
signature	$y = sig_k(z)$	$y \in Y$

Figure 3.1 Signing a message digest

3.3 COLLISION-FREE HASH FUNCTIONS

We have to be careful that the use of a hash function h does not weaken the security of the signature scheme, for it is the message digest that is signed, not the message. It will be necessary for h to satisfy certain properties in order to prevent various forgeries.

The most obvious type of attack is for an opponent, Oscar, to start with a valid signed message (x, y), where $y = sig\ K\ (h(x))$. (The pair (x, y) could be any message previously signed by Bob.) Then he computes $z = h(x)$ and attempts to find $x' \neq x$ such that $h(x') = h(x)$. If Oscar can do this, (x', y) would be a valid signed message, i.e., a *forgery*. In order to prevent this type of attack, we require that h satisfy the following collision-free property:

DEFINITION:

Let x be a message. A hash function h is weakly collision-free for x if it is computationally infeasible to find a message $x' \neq x$ such that $h(x') = h(x)$.

Another possible attack is the following: Oscar first finds two messages $x \neq x'$ such that $h(x) = h(x')$. Oscar then gives x to Bob and persuades him to sign the message digest $h(x)$, obtaining y. Then (x', y) is a valid forgery. This motivates a different collision-free property:

DEFINITION

A hash function h is strongly collision-free if it is computationally infeasible to find messages x and x′ such that $x' \neq x$ and $h(x') = h(x)$.

Observe that a hash function h is strongly collision-free if and only if it in computationally infeasible to find a message x such that h is not weakly collision-free for x. Here is a third variety of attack. As we mentioned in Section it is often possible with certain signature schemes to

forge signatures on random message digests *z*. Suppose Oscar computes a signature on such a random *z*, and then he finds a message *x* such that *z* = *h*(*x*). If he can do this, then (*x*, *y*) is a valid forgery. To prevent this attack, we desire that *h* satisfy the same one-way property that was mentioned previously in the context of public-key cryptosystems and the **Lamport Signature Scheme**:

DEFINITION

A hash function h is one-way if, given a message digest z, it is computationally infeasible to find a message x such that h(*x*) = *z*.

We are now going to prove that the strongly collision-free property implies the one-way property. This is done by proving the contra positive statement. More specifically, we will prove that an arbitrary inversion algorithm for a hash function can be used as an oracle in a Las Vegas probabilistic algorithm that finds collisions.

This reduction can be accomplished with a fairly weak assumption on the relative sizes of the domain and range of the hash function. We will assume for the time being that the hash function $h : X \tilde{O} Z$, where X and Z are finite sets and $|X| \geq 2|Z|$. This is a reasonable assumption: If we think of an element of X as being encoded as a bitstring of length $\log 2\ |X|$ and an element of Z as being encoded as a bitstring of length $\log 2\ |Z|$, then the message digest $z = h(x)$ is at least one bit shorter than the message x.

(Eventually, we will be interested in the situation where the message domain X is infinite, since we want to be able to deal with messages of arbitrary length. Our argument also applies in this situation.)

We are assuming that we have an inversion algorithm for *h*. That is, we have an algorithm **A** which accepts as input a message digest $z \in Z$, and finds an element $\mathbf{A}(z) \in X$ such that $h(\mathbf{A}(z)) = z$.

Suppose $h: X \to Z$ is a hash function where $|X|$ and $|Z|$ are finite and $|X| \geq 2|Z|$. Suppose a is an inversion algorithm for h. Then there exists a probabilistic Las Vegas algorithm which finds a collision for h with probability at leats 1/2 .

Proof consider the algorithm presented in figure 7.2 . Clearly B is a probabilistic algorithm of the Las Vegas type, since it either finds a collision or returns no answer. Thus our main task is to compute the probability of success. For any $x \in X$, define $x \sim x_1$ if h(x)=h(x_1). It is easy to see that $\sim$ is an equivalence relation. Define

$$[x] = \{x_1 \in X : x \sim x_1\}.$$

Each equivalence class $[x]$ consist of the inverse image of an element of Z, so the number of equivalence classes is at most $[z]$. Denote the set of equivalence classes by C.

Now suppose x is the element of X chosen step 1. For this x, there are $\|[x]\|$ possible x_1's that could be returned in step 3. $\|[x]\|$-1of these x_1's are different from x and thus lead to success in step 4. (Note that the algorithm A dose not know the representative of the equivalence class $[x]$ that was chosen in step 1)

So, given a particular choice $x \in X$, the probability of success is $(\|[x]\|-1)/\|[x]\|$.

3.4 THE BIRTHDAY ATTACK

In this section, we determine a necessary security condition for hash functions that depends only on the cardinality of the set *Z* (equivalently, on the size of the message digest). This necessary condition results from a simple method of finding collisions which is informally known as the *birthday attack*. This terminology arises from the so-called *birthday paradox*, which says that in a group of 23 random people, at least two will share a birthday with probability at least 1/2. (Of course this is not a paradox, but it is probably counter-intuitive). The reason for the terminology "birthday attack" will become clear as we progress.

As before, let us suppose that $h : X \tilde{O} Z$ is a hash function, *X* and *Z* are finite, and $|X| \geq 2|Z|$. Denote $|X| = m$ and $|Z| = n$. It is not hard to see that there are at least *n* collisions — the question is how to find them.

A very naive approach is to choose *k* random distinct elements $x1,...,x\,k \in X$, compute $zi = h(xi)$, $1 \leq i \leq k$, and then determine if a collision has taken place (by sorting the *zi* 's, for example).
This process is analogous to throwing *k* balls randomly into *n* bins and then checking to see if some bin contains at least two balls. (The *k* balls correspond to the *k* random *xi*'s, and the *n* bins correspond to the *n* possible elements of *Z*.)

We will compute a lower bound on the probability of finding a collision by this method. This lower bound will depend on *k* and *n*, but not on *m*. Since we are interested in a lower bound on the collision probability, we will make the assumption that for all $z \in Z$. (This is a reasonable assumption: if the inverse images are not approximately equal, then the probability of finding a collision will increase.)

Since the inverse images are all (roughly) the same size and the *xi*'s are chosen at random, the resulting *zi*'s can be thought of as random (not necessarily distinct) elements of *Z*. But it is a simple matter to compute the probability that *k* random elements *z*1,...,*z k* ∈ *Z* are distinct. Consider the *zi*'s in the order *z1*,...,*z k*. The first choice *z*1 is arbitrary; the probability that *z*2 *z*1is 1 - 1/*n*; the probability that *z*3isdistinct from z1 and *z*2 is 1 - 2/*n*, etc.

Hence, we estimate the probability of no collisions to be

$$\left(1-\frac{1}{n}\right)\left(1-\frac{2}{n}\right).....\left(1-\frac{k-1}{n}\right)=\prod_{i=1}^{k-1}\left(1-\frac{i}{n}\right)$$

If x is a small real number, then 1-1$-x \approx e^{-x}$ This estimate is derived by taking the first two terms of the series expansion.

$$e^{-x}=1-x+\frac{x^2}{2!}-\frac{x^3}{3!}.....$$

Then our estimated probability of no collisions is

$$\prod_{i=1}^{k-1}\left(1-\frac{i}{n}\right)\approx\prod_{i=1}^{k-1}e^{\frac{-1}{n}}$$

$$=e^{\frac{-k(k-1)}{2n}}$$

So we estimated probability of at least one collisions to be

$$1-e^{\frac{-k(k-1)}{2n}}$$

If we denote this probability by $\in$then we can solve for k as an function of n and $\in$

$$=e^{\frac{-k(k-1)}{2n}}\approx 1-c$$

$$\frac{-k(k-1)}{2n}\approx 1n(1-\in)$$

$$k^2-k\approx 2n1n\frac{1}{1-\in}$$

If we ignore the term –k, then we estimate

$$k\approx\sqrt{2n1n\frac{1}{1-\in}}$$

If we take $\in$=5 then our estimate is

$$k \approx 1.17\sqrt{n}.$$

So this says that hashing just over random elements of X yields a collision with a probability of 50%. Note that a different choice of $\in$ leads to a different constant factor, but k will still be proportional to If X is the set of all human beings, Y is the set of 365 days in a non-leap year (i.e., excluding February 29), and $h(x)$ denotes the birthday of person x, then we are dealing with the birthday paradox. Taking n = 365 in our estimate, we get Hence, as mentioned earlier, there will be at least one duplicated birthday among 23 random people with probability at least 1/2.

3.5 A DISCRETE LOG HASH FUNCTION

In this section, we describe a hash function, due to Chaum, van Heijst, and Pfitzmann, that will be secure provided a particular discrete logarithm cannot be computed. This hash function is not fast enough to be of practical use, but it is conceptually simple and provides a nice example of a hash function that can be proved secure under a reasonable computational assumption. The **Chaum-van Heijst-Pfitzmann Hash Function** is presented. We now prove a theorem concerning the security of this hash function.

THEOREM

Given one collision for the **Chaum-van Heijst-Pfitzmann Hash Function** *h, the discrete logarithm* $\log_\alpha \beta$ *can be computed efficiently*.

PROOF Suppose we are given a collision **h(x1,x2)=h(x3,x4)** where ($x1$, $x2$) ≠ ($x3$, $x4$). So we have the following congruence:

PROOF Suppose we are given a collision **h(x1,x2)=h(x3,x4)** where ($x1$, $x2$) ≠ ($x3$, $x4$). So we have the following congruence:

$$\sim^{x_1} {}^{x_2} \equiv \sim^{x_3} {}^{x_4} \pmod{p},$$

Or

$$\sim^{x_1 - x_3} \equiv {}^{x_4 - x_2} \pmod{p},$$

Denote

$$d = \gcd(x_4 - x_2, p - 1).$$

Since p - 1 = 2q and q is prime, it must be the case that d ∈ {1,2,q,p - 1}. Hence, we have four possibilities for d, which we will consider in turn. First, suppose that d = 1. Then let

$$y = (x_4 - x_2)^{-1} \bmod (p-1).$$

We have that

$$\equiv \quad ^{(x_4 - x_2)y} \pmod{p}.$$

$$\equiv \sim ^{(x_1 - x_3)y} \pmod{p}.$$

So we can compute the discrete logarithm $\log_{\sim}$ as follows

$$\log_a \quad \equiv (x_1 - x_3)(x_4 - x_2)^{-1} \bmod (p-1).$$

Next suppose that d=2 Since p-1=2q where q is odd we must have gcd (x_4-x_2, q)=1 Let

$$y = (x_4 - x_2)^{-1} \bmod q.$$

Now

$$(x_4 - x_2)y = kq + 1$$

For some integer k, so we have

$$(x_4 - x_2)y \equiv \quad ^{kq+1} \pmod{p}$$

$$\equiv (-1) \quad \pmod{p}$$

$$\pm \quad \pmod{p}$$

Since

$$^{q} = -1 \pmod{p}$$

so we have

$$^{(x_4 - x_2)} \equiv \sim ^{(x_1 - x_3)y} \pmod{p}$$

$$\equiv \pm \quad \pmod{p}$$

It follows that

$$\log_{\sim} \quad = (x_1 - x_3)\, y \bmod (p-1)$$

Or

$$\log_{\sim} \quad = (x_1 - x_3)\, y + q \bmod (p-1)$$

We can easily test which of these two possibilities is the correct one. Hence, as in the case $d = 1$, we have calculated the discrete logarithm $\log_{\alpha} \beta$.The next possibility is that $d = q$. Buts

$$0 \le x_2 \le q-1$$

And

$$0 \le x_4 \le q-1$$

So

$$-(q-1) \le x_4 - x_2 \le q-1$$

So it is impossible that gcd(*x*4- *x*2,*p* - 1) = *q*; in other words, this case does not arise. The final possibility is that *d* = *p* - 1. This happens only if *x*2= *x*4. But then we have and *x*1= *x*3.

$$\sim^{x_1 \quad x_2} \equiv \sim^{x_3 \quad x_2} (\mathrm{mod}\ p)$$

So

$$\sim^{x_1} \equiv \sim^{x_3} (\mathrm{mod}\ p)$$

Thus (*x*2, *x*2) = (*x*3, *x*4), a contradiction. So this case is not possible, either. Since we have considered all possible values for *d*, we conclude that the hash function *h* is strongly collision-free provided that it is infeasible to compute the discrete logarithm log α β in Z_p. We illustrate the result of the above theorem with an example.

Suppose P=12347(SO Q= 6173), $\sim = 2 \quad = 8461$ Suppose we are given the collision

$$\sim^{5692 \quad 144} \equiv \sim^{212 \quad 4214} (\mathrm{mod} 12347)$$

Thus

$$\sim^{5692 \quad 144} \equiv \sim^{212 \quad 4214} (\mathrm{mod} 12347)$$

Thus x_1=5692, x_2=144, x_3=212 and x_4=4214. Now, gcd(x_4-x_2, p-1)=2, so we begin by computing

$$y = (x_4 - x_2)^{-1} \bmod q.$$
$$= (4214 - 144)^{-1} \bmod 6173.$$
$$= 4312$$

Next we compute

$$y' = (x_1 - x_3)\, y \bmod (p-1).$$
$$= (5692 - 212)\, 4312 \bmod 12346$$
$$= 11862$$

Now it is the case that $\log_{\sim} \quad \in \{y', y' + q \bmod (p-1)\}$ since

$\sim^{y'} \bmod p = 2^{11862} \bmod 12346 = 9998$

We conclude that

$$\log_{\sim} \quad = y' + q \bmod (p-1)$$

$$11862 + 6173 \bmod 12346$$
$$= 5689$$

As We check, we can verify that

$$2^{5689} \equiv 8461 (\bmod 12347)$$

Hence we have determined $\log_{\sim}$

3.6 EXTENDING HASH FUNCTIONS

So far, we have considered hash functions with a finite domain. We now study how a strongly collision free hash function with a finite domain can be extended to a strongly collision-free hash function with an infinite domain. This will enable us to sign messages of arbitrary length.

Suppose $h : (\mathbb{Z}_2)^m \to (\mathbb{Z}_2)^t$ is a strongly collision-free hash function, where $m \geq t + 1$. We will use h to construct a strongly collision-free hash function $h^* : X \to (\mathbb{Z}_2)^t$, where

$$X = \bigcup_{i=m}^{\infty} (\mathbb{Z}_2)^i$$

We first consider the situation where $m \geq t + 2$. We will think of elements of X as bit-strings. $|x|$ denotes the length of x (i.e., the number of bits in x), and $x \| y$ denotes the concatenation of the bit-strings x and y. Suppose $|x| = n > m$. We can express x as the concatenation

$$x = x_1 \| x_2 \| ..... \| x_k$$

Where

$$|x_1| = |x_2| = ... = |x_{k-1}| = m - t - 1$$

And

$$|x_k| = m - t - 1 - d$$

Where $0 \leq d \leq m - t - 2$ hence we have that

$$k = \left\lceil \frac{n}{m - t - 1} \right\rceil$$

We define h* x by algorithm presented in fig 2.4

3.7 HASH FUNCTIONS FROM CRYPTOSYSTEMS

So far, the methods we have described lead to hash functions that are probably too slow to be useful in practice. Another approach is to use an existing private-key cryptosystem to construct a hash function.

Let us suppose that ***(P,C,K,E,D)***is a computationally secure cryptosystem. For convenience, let us assume also that P=C=K=(Z_2) ^n. Here we should have $n \geq 128$, say, in order to prevent birthday attacks. This precludes using **DES** (as does the fact that the key length of **DES** is different from the plaintext length).

Suppose we are given a bitstring

$$x = x_1 \| x_2 \| ..... \| x_k$$

Where $x_i \in (\mathbb{Z}_2)^n, 1 \leq i \leq k$ (if the number of bits in x is not a multiplr of n, then it will be necessary to pad x in some way, such as was done in Section 2.5. for simplicity, we will ignore this now).

The basic idea to begin with a fixed 'initial value' g_0=iv and then construct $g_1 \ldots . g_k$ in order by a rule of the form

$$g_i = f(x_i, g_{i-1})$$

Where f is a function that incorporates the encryption of our cryptosystem. Finally define the message digest h(x)=g_k

Several hash functions of this type have been proposed, and many of them have been shown to be insecure (independent of whether or not the underlying cryptosystem is secure). However, four variations of this theme that appear to be secure are as follows:

$$g_i = e_{gi-1}(x_i) \oplus x_i$$
$$g_i = e_{gi-1}(x_i) \oplus x_i \oplus g_{i-1}$$
$$g_i = e_{gi-1}(x_i \oplus g_{i-1}) \oplus x_i$$
$$g_i = e_{gi-1}(x_i \oplus g_{i-1}) \oplus x_i \oplus g_{i-1}$$

3.8 THE MD4 HASH FUNCTION

The **MD4 Hash Function** was proposed in 1990 by Rivest, and a strengthened version, called **MD5**, was presented in 1991. The **Secure Hash Standard** (or **SHS**) is more complicated, but it is based on the same underlying methods. It was published in the Federal Register on January 31, 1992, and adopted as a standard on May 11, 1993. (A proposed revision was put forward on July 11, 1994, to correct a "technical flaw" in

the **SHS**.) All of the above hash functions are very fast, so they are practical for signing very long messages. In this section, we will describe **MD4** in detail, and discuss some of the modifications that are employed in **MD5** and the **SHS**.

Given a bitstring *x*, we will first produce an array

$$M = M[0]M[1].....M[N-1]$$

where each $M[i]$ is a bitstring of length 32 and $N \equiv 0 \bmod 16$. We will call each $M[i]$ a *word*. *M* is constructed from *x* using the algorithm presented. In the construction of *M*, we append a single 1 to *x*, then we concatenate enough 0's so that the length becomes congruent to 448 modulo 512, and finally we concatenate 64 bits that contain the binary representation of the (original) length of *x* (reduced modulo 2^{64}, if necessary). The resulting string *M* has length divisible by 512. So when we break *M* up into 32-bit words, the resulting number of words, denoted by *N*, will be divisible by 16.

Now we proceed to construct a 128-bit message digest. A high-level description of the algorithm is presented. The message digest is constructed as the concatenation of the four words *A*, *B*, *C* and *D*, which we refer to as *registers*. The four registers are initialized in step 1. Now we process the array *M* 16 words at a time. In each iteration of the loop in step 2, we first take the "next" 16 words of *M* and store them in an array *X* (step 3). The values of the four register are then stored (step 4). Then we perform three "rounds" of hashing. Each round consists of one operation on each of the 16 words in *X* (we will describe these operations in more detail shortly). The operations done in the three rounds produce new values in the four registers. Finally, the four registers are updated in step 8 by adding back the values that were stored in step 4. This addition is defined to be addition of positive integers, reduced modulo 2^{32}.

The three rounds in **MD4** are different (unlike **DES**, say, where the 16 rounds are identical). We first describe several different operations that are employed in these three rounds. In the following description, *X* and *Y* denote input words, and each operation produces a word as output. Here are the operations employed:

$X \wedge Y$	bitwise ''and '' of X and Y
$X \vee Y$	bitwise ''OR'' of X and Y
$X \oplus Y$	bitwise ''XOR'' of X and Y
$\overline{X}$	bitwise complement of X
$X + Y$	integer addition modulo 2^{32}
$X \lll s$	circular left shift of s positions $(0 \le s \le 31)$

Note that all of these operations are very fast, and the only arithmetic operation that is used is addition modulo 2^{32}. If **MD4** is actually

implemented, it will be necessary to take into account the underlying architecture of the computer it is run on in order to perform addition correctly. Suppose $a_1a_2a_3a_4$ are the four bytes in a word. We think of each a_i as being an integer in the range 0,...,255, represented in binary.

In a *big-endian* architecture (such as a Sun SPARCstation), this word represents the integer

$$a_1 2^{24} + a_2 2^{16} + a_3 2^8 + a_4$$

In a little –endian architecture(such as the Intel 80xxx line), this word represents the integer

$$a_4 2^{24} + a_3 2^{16} + a_2 2^8 + a_1$$

MD4 assumes a little-endian architecture. It is important that the message digest is independent of the underlying architecture. So if we wish to run **MD4** on a big-endian computer, it will be necessary to perform the addition operation $X + Y$ as follows:

1. Interchange x_1 and x_4;x_2 and x_3;y_1 and y_4; and y_2 and y_3.
2. Compute $Z=X+Y \bmod 2^{32}$
3. Interchange z_1 and z_4; and z_2 and z_3.

Rounds 1, 2, and 3 of **MD4** respectively use three functions *f*, *g* and *h*. Each of *f*, *g* and *h* is a bitwise boolean function that takes three words as input and produces a word as output. They are defined as follows:

$$f(X,Y,Z) = (X \wedge Y) \vee ((\neg X) \wedge Z)$$
$$g(X,Y,Z) = (X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z)$$
$$h(X,Y,Z) = X \oplus Y \oplus Z$$

The complete description of Rounds 1, 2 and 3 of **MD4** was designed to be very fast, and indeed, software implementations on Sun SPARCstations attain speeds of 1.4 Mbytes/sec. On the other hand, it is difficult to say something concrete about the security of a hash function such as **MD4** since it is not "based" on a well-studied problem such as factoring or the **Discrete Log** problem. So, as is the case with **DES**, confidence in the security of the system can only be attained over time, as the system is studied and (one hopes) not found to be insecure.

Although **MD4** has not been broken, weakened versions that omit either the first or the third round can be broken without much difficulty. That is, it is easy to find collisions for these two-round versions of **MD4**. A strengthened version of **MD4**, called **MD5**, was proposed in 1991. **MD5**

uses four rounds instead of three, and runs about 30% slower than **MD4** (about .9 Mbytes/sec on a SPARCstation).

3.9 TIME STAMPING

One difficulty with signature schemes is that a signing algorithm may be compromised. For example, suppose that Oscar is able to determine Bob's secret exponent *a* in the **DSS**. Then, of course, Oscar can forge Bob's signature on any message he likes. But another (perhaps even more serious) problem is that the compromise of a signing algorithm calls in to question the authenticity of all messages signed by Bob, including those he signed before Oscar stole the signing algorithm.

Here is yet another undesirable situation that could arise: Suppose Bob signs a message and later wishes to disavow it. Bob might publish his signing algorithm and then claim that his signature on the message in question is a forgery. The reason these types of events can occur is that there is no way to determine when a message was signed. This suggests that we consider ways of *timestamping* a (signed) message. A timestamp should provide proof that a message was signed at a particular time. Then, if Bob's signing algorithm is compromised, it would not invalidate any signatures he made previously. This is similar conceptually to the way credit cards work: if someone loses a credit card and notifies the bank that issued it, it becomes invalid. But purchases made prior to the loss of the card are not affected.

In this section, we will describe a few methods of timestamping. First, we observe that Bob can produce a convincing timestamp on his own. First, Bob obtains some "current" publicly available information which could not have been predicted before it happened. For example, such information might consist of all the major league baseball scores from the previous day, or the values of all the stocks listed on the New York Stock Exchange. Denote this information by *pub*.

Now, suppose Bob wants to timestamp his signature on a message *x*. We assume that *h* is a publicly known hash function. Bob will proceed according to the algorithm presented. Here is how the scheme works: The presence of the information *pub* means that Bob could not have produced *y* before the date in question. And the fact that *y* is published in the next day's newspaper proves that Bob did not compute *y* after the date in question. So Bob's signature *y* is bounded within a period of one day.

Also observe that Bob does not reveal the message *x* in this scheme since only *z* is published. If necessary, Bob can prove that *x* was the message he signed and timestamped simply by revealing it.

It is also straightforward to produce timestamps if there is a trusted timestamping service available (i.e., an electronic notary public). Bob can compute $z = h(x)$ and $y = sig_K(z)$ and then send (z, y) to the timestamping

service, or TSS. The TSS will then append the date *D* and sign the triple (*z*, *y*, *D*).

This works perfectly well provided that the signing algorithm of the TSS remains secure and provided that the TSS cannot be bribed to backdate timestamps. (Note also that this method establishes only that Bob signed a message before a certain time. If Bob also wanted to establish that he signed it after a certain date, he could incorporate some public information *pub* as in the previous method.)

If it is undesirable to trust the TSS unconditionally, the security can be increased by sequentially linking the messages that are timestamped. In such a scheme, Bob would send an ordered triple (*z, y*, ID(Bob)) to the TSS. Here *z* is the message digest of the message *x*; *y* is Bob's signature on *z*; and ID(Bob) is

Bob's identifying information. The TSS will be timestamping a sequence of triples of this form. Denote by (*zn*, *yn*, ID*n*) the *n*th triple to be timestamped by the TSS, and let *tn* denote the time at which the *n*th request is made.

The TSS will timestamp the *n*th triple using the algorithm. The quantity *Ln* is "linking information" that ties the *n*th request to the previous one. (*L*0 will be taken to be some predetermined dummy information to get the process started.)

Now, if challenged, Bob can reveal his message *xn*, and then *yn* can be verified. Next, the signature *sn* of the TSS can be verified. If desired, then ID *n*-1 or ID *n*+1 can be requested to produce their timestamps, (*C n*-1 , *sn*-1, ID*n*) and (*Cn*+1, *sn*+1, ID*n*+2), respectively. The signatures of the TSS can be checked in these timestamps. Of course, this process can be continued as far as desired, backwards and/or forwards.

3.10 LET US SUM UP

Thus, we have studied the basic concepts about the structure cryptographic hash function. Here, message digest generation MD4 and MD5 and Birthday attack explained briefly. The working of time stamping also explained.

3.11 LIST OF REFERENCES

- **Cryptography: Theory and Practice,** *Douglas Stinson*, CRC Press, CRC Press LLC
- **Cryptography and Network Security Principles and Practices**, Fourth Edition, William Stallings, PHI(Pearson),

3.12 UNIT END EXERCISES

1. Explain the MD4 algorithm for message digest generation.
2. What are the different cryptographic hash function criteria?
3. Explain Birthday Attack in detail.
4. Explain working of time stamping with application.
5. Describe Hash function.

*****

UNIT-II

4

KEY DISTRIBUTION AND KEY AGREEMENT

Unit Structure

4.0 OBJECTIVES

In this chapter you will learn about:

- What is key predistribution?
- Diffie-Hellman key predistribution.
- What is Kerberos?
- The station to station protocol and MIT key agreement protocol
- Key agreement using self-certifying keys

4.1 INTRODUCTION

We have observed that public-key systems have the advantage over private-key systems that a secure channel is not needed to exchange a secret key. But, unfortunately, most public-key systems are much slower than private-key systems such as **DES**, for example. So, in practice, private-key systems are usually used to encrypt "long" messages. But then we come back to the problem of exchanging secret keys.

In this chapter, we discuss several approaches to the problem of establishing secret keys. We will distinguish between key distribution and

key agreement. *Key distribution* is defined to be a mechanism whereby one party chooses a secret key and then transmits it to another party or parties. *Key agreement* denotes a protocol whereby two (or more) parties jointly establish a secret key by communicating over a public channel. In a key agreement scheme, the value of the key is determined as a function of inputs provided by both parties.

As our setting, we have an insecure network of *n* users. In some of our schemes, we will have a *trusted authority* (denoted by TA) that is responsible for such things as verifying the identities of users, choosing and transmitting keys to users, etc.

Since the network is insecure, we need to protect against potential opponents. Our opponent, Oscar, might be a *passive adversary*, which means that his actions are restricted to eavesdropping on messages that are transmitted over the channel. On the other hand, we might want to guard against the possibility that Oscar is an *active adversary*. An active adversary can do various types of nasty things such as the following:

1. alter messages that he observes being transmitted over the network
2. save messages for reuse at a later time
3. attempt to masquerade as various users in the network.

The objective of an active adversary might be one of the following:

1. to fool U and V into accepting an "invalid" key as valid (an invalid key could be an old key that has expired, or a key chosen by the adversary, to mention two possibilities)
2. to make U or V believe that they have exchanged a key with other when they have not.

The objective of a key distribution or key agreement protocol is that, at the end of the protocol, the two parties involved both have possession of the same key *K*, and the value of *K* is not known to any other party (except possibly the TA). Certainly it is much more difficult to design a protocol providing this type of security in the presence of an active adversary as opposed to a passive one. We first consider the idea of *key predistribution* in Section. For every pair of users {U, V}, the TA chooses a random key *K*U,V= *K*V,U and transmits it "off-band" to U and V over a secure channel. (That is, the transmission of keys does not take place over the network, since the network is not secure.) This approach is unconditionally secure, but it requires a secure channel between the TA and every user in the network. But, of possibly even more significance is the fact that each user must store *n* - 1 keys, and the TA needs to transmit a total of ($\frac{n}{2}$) keys securely (this is sometimes called the "*n*2 problem"). Even for relatively small networks, this can become prohibitively expensive, and thus it is not really a practical solution.

In Section, we discuss an interesting unconditionally secure key predistribution scheme, due to Blom, that allows a reduction in the amount of secret information to be stored by the users in the network. We also present in Section a computationally secure key predistribution scheme based on the discrete logarithm problem. A more practical approach can be described as *on-line key distribution by TA*. In such a scheme, the TA acts as a *key server*. The TA shares a secret key *K* U with every user U in the network. When U wishes to communicate with V, she requests a *session key* from the TA. The TA generates a session key *K* and sends it in encrypted form for U and V to decrypt. The well-known **Kerberos** system, which we describe in Section 8.3, is based on this approach.

If it is impractical or undesirable to have an on-line TA, then a common approach is to use a *key agreement protocol*. In a key agreement protocol, U and V jointly choose a key by communicating over a public channel. This remarkable idea is due to Diffie and Hellman, and (independently) to Merkle. We describe a few of the more popular key agreement protocols. A variation of the original protocol of Diffie and Hellman, modified to protect against an active adversary, is presented in Section. Two other interesting protocols are also discussed: the **MTI** scheme is presented in Section and the **Girault** scheme is covered in Section.

4.2 KEY PREDISTRIBUTION

In the basic method, the TA generates keys, and gives each key to a unique pair of users in a network of *n* users. As mentioned above, we require a secure channel between the TA and each user to transmit these keys. This is a significant improvement over each pair of users independently exchanging keys over a secure channel, since the number of secure channels required has been reduced from to *n*. But if *n* is large, this solution is not very practical, both in terms of the amount of information to be transmitted securely, and in the amount of information that each user must store securely (namely, the secret keys of the other other *n* - 1 users). Thus, it is of interest to try to reduce the amount of information that needs to be transmitted and stored, while still allowing each pair of users U and V to be able to (independently) compute a secret key *K*U,V. An elegant scheme to accomplish this, called the **Blom Key Predistribution Scheme.**

4.3 BLOM'S SCHEME

As above, we suppose that we have a network of *n* users. For convenience, we suppose that keys are chosen from a finite field Z_p , where $p \geq n$ is prime. Let *k* be an integer, $1 \geq k \geq n - 2$. The value *k* is the largest size coalition against which the scheme will remain secure. In the **Blom Scheme**, the TA will transmit $k + 1$ elements of Z_p to each user over a secure channel (as opposed to $n - 1$ in the basic key predistribution

scheme). Each pair of users, U and V, will be able to compute a key *K*U,V= *K*V,U , as before. The security condition is as follows: any set of at most *k* users disjoint from {U, V} must be unable to determine any information about *K*U,V (note that we are speaking here about unconditional security).

We first present the special case of Blom's scheme where $k = 1$. Here, the TA will transmit two elements of to each user over a secure channel, and any individual user W will be unable to determine any information about *K*U,V if W $\neq$ U, V. We illustrate the **Blom Scheme** with $k = 1$ in the following example.

Example 4.1:

Suppose the three users are U,V AND W, p=17 and their public elements are r_u =12, r_v =7 and r_w=

1. Suppose that the TA chooses a=8,b=7 and c=2, so the polynomial *f* is

$$f(x, y) = 8 + 7(x + y) + 2xy.$$

The g polynomials are as follows

$$\mathrm{gu}(x) = 7 + 14x$$
$$\mathrm{gv}(x) = 6 + 4x$$
$$gw(x) = 15 + 9x$$

The three keys are thus

$$Ku, v = 3$$
$$\mathrm{K_{U,W}} = 4$$
$$\mathrm{K_{V,W}} = 10$$

U would complete k_{UV} as

$$gu(rv) = 7 + 14 \times 7 \bmod 17 = 3$$

V would complete k_{UV} as

$$gv(ru) = 6 + 4 \times 12 \bmod 17 = 3$$

We leave the computation of the other keys as an exercise for the reader. We now prove that no one user can determine any information about the key of two other users.

THEOREM

The **Blom Scheme** *with* $k = 1$ *is unconditionally secure against any individual user.* **PROOF** Let's suppose that user W wants to try to compute the key

$$Kuv = a + b(ru + rv) + crurv \bmod p.$$

The values ru, rv are public but a,b and are unknown W does know the values.

$$aw = a + brw \bmod p.$$

And

$$bw = b + crw \bmod p.$$

since these are the coefficients of the polynomial gW (x) that was sent to W by the TA.

What we will do is show that the information known by W is consistent with any possible value $l \in \mathbb{Z}p.$ of the key K_{uv}. Hence , W cannot rule out any values for K_{uv}. Condsider the following matrix equation (in $\mathbb{Z}p.$)

$$\begin{pmatrix} 1 & ru+rv & rurv \\ 1 & rw & 0 \\ 0 & 1 & rw \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} l \\ aw \\ bw \end{pmatrix}$$

The first equation represents the hypothesis that Kuv=l the second and third equations contain the information that W knows about a, b and c form g_w(x).

The determinants of the coefficient matrix is

$$rw^2 + rurv - (ru+rv)rw = (rw-ru)(rw-rv),$$

where all arithmetic is done in $\mathbb{Z}p.$. Since r $r_w \neq r_u$ and $r_w \neq r_v$, it follows that the coefficient matrix has non zero determinant and hence the matrix equation has a unique solution for a,b,c. In other words, any possible value l of kuv…. Is consistent with the information known to W

On the other hand ,a coalition of two users, say{W,X}, will be able to determine any key K_{UV} where {W,X} $\{W, X\} \cap (U, \mathrm{V}) = \emptyset$ WandX together know that

o

$$aw = a + brw.$$

$$bw = b + crw.$$

$$a_X = a + brx$$

$$bx = b + crx.$$

Thus they have four equations in three unknowns, and they can easily compute a unique solution for *a, b* and *c*. Once they know *a*, *b* and *c*, they can form the polynomial *f*(*x, y*) and compute any key they wish. It

is straightforward to generalize the scheme to remain secure against coalitions of size *k*. The only thing that changes is step 2. The TA will use a polynomial *f*(*x*, *y*) having the form

$$f(x,y)=\sum_{i=0}^{k}\sum_{j=0}^{k}a_{i,j}x^{i}y^{j} \bmod p$$

Where $a_{i,j}\in\mathbb{Z}_p\,(0\le i\le k,0\le j\le k)$ and $a_{i,j}=a_{j,i}$ the remainder of the protocol is unchanged

4.4 DIFFIE-HELLMAN KEY PREDISTRIBUTION

In this section, we describe a key predistribution scheme that is a modification of the well-known Diffie-Hellman key exchange protocol that we will discuss a bit later, in Section 8.4. We call this the **Diffie-Hellman Key Predistribution Scheme**. The scheme is computationally secure provided a problem related to the **Discrete Logarithm** problem is intractible. We will describe the scheme Z_p over , where p is prime, though it can be implemented in any finite group in which the **Discrete Logarithm** problem is intractible. We will assume that a is a primitive element of Z_p, and that the values *p* and α are publicly known to everyone in the network. In this scheme, ID(U) will denote certain identification information for each user U in the network, e.g., his or her name, e-mail address, telephone number, or other relevant information. Also, each user U has a secret exponent *a*U (where 0 ≤ *a* U ≤ *p* - 2), and a corresponding public value

$$b_U = \sim^{av} \bmod p$$

The TA will have a signature scheme with a (public) verification algorithm *ver* TA and a secret signing algorithm *sig*TA. Finally, we will implicitly assume that all information is hashed, using a public hash function, before it is signed. To make the procedures easier to read, we will not include the necessary hashing in the description of the protocols.Certain information pertaining to a user U will be authenticated by means of a *certificate* which is issue and signed by the TA. Each user U will have a certificate

$$C(U)=\left(ID(U),b_U,sig_{TA}\left(ID(U),b_U\right)\right),$$

where *b*U is formed as described above (note that the TA does not need to know the value of *a*U). A certificate for a user U will be issued when U joins the network. Certificates can be stored in a public database, or each user can store his or her own certificate. The signature of the TA on a certificate allows anyone in the network to verify the information it contains. It is very easy for U and V to compute the common key

$$K_{U,V} = \sim^{a_U a_V} \bmod P,$$

4.5 KERBEROS

In the key predistribution methods we discussed in the previous section, each pair of users can compute one fixed key. If the same key is used for a long period of time, there is a danger that it might be compromised. Thus it is often preferable to use an on-line method in which a new session key is produced every time a pair of users want to communicate (this property is called *key freshness*). If on-line key distribution is used, there is no need for any network user to store keys to communicate with other users (each user will share a key with the TA, however). Session keys will be transmitted on request by the TA. It is the responsibility of the TA to ensure key freshness. **Kerberos** is a popular key serving system based on private-key cryptography. In this section, we give an overview of the protocol for issuing session keys in **Kerberos**. Each user U shares a secret **DES** key K_U with the TA. In the most recent version of **Kerberos** (version V), all messages to be transmitted are encrypted using cipher block chaining (CBC) mode, as described in Section.

As in Section, ID(U) will denote public identification information for user U. When a request for a session key is sent to the TA, the TA will generate a new random session key K. Also, the TA will record the time at which the request is made as a *timestamp*, *T*, and specify the *lifetime*, *L*, during which *K* will be valid. That is, the session key *K* is to be regarded as a valid key from time *T* to time *T* + *L*. All this information is encrypted and transmitted to U and (eventually) to V. Before going into more details, we will present the protocol

The information transmitted in the protocol is illustrated in the following diagram:

$$\begin{array}{ccccc} & eku(\mathrm{K,ID(V),T,L}) & & eku(\mathrm{ID(U),T}) & \\ \mathrm{TA} & ek_V(\mathrm{K,ID(U),T,L}) & \mathrm{U} & ek_V(\mathrm{K,ID(U),T,L}) & \mathrm{V} \\ \longrightarrow & & & \longrightarrow & \\ & & & ek(T+1) & \\ & & & \longleftarrow & \end{array}$$

We will now explain what is going on in the various steps of the protocol. Although we have no formal proof that **Kerberos** is "secure" against an active adversary, we can at least give some informal motivation of the features of the protocol.

As mentioned above, the TA generates *K*, *T*, and *L* in step 2. In step 3, this information, along with ID (V), is encrypted using the key K_U shared by U and the TA to form m_1. Also, *K*, *T*, *L*, and ID(U) are encrypted using the key K_V shared by V and the TA to form m_2. Both these encrypted messages are sent to U. U can use her key to decrypt m_1, and thus obtain *K*, *T*, and *L*. She will verify that the current time is in the

interval from T to $T + L$. She can also check that the session key K has been issued for her desired communicant V by verifying the information ID(V) decrypted from m_1.

Next, U will relay m_2 to V. As well, U will use the new session key K to encrypt T and ID (U) and send the resulting message m_3 to V. When V receives m_2 and m_3 from U, he decrypts m_2 to obtain T, K, L and ID(U). Then he uses the new session key K to decrypt m_3 and he verifies that T and ID(U), as decrypted from m_2 and m_3, are the same.

This ensures V that the session key encrypted within m_2 is the same key that was used to encrypt m_3.Then V uses K to encrypt $T + 1$, and sends the result back to U as message m_4.When U receives m_4, she decrypts it using K and verifies that the result is $T + 1$. This ensures U that the session key K has been successfully transmitted to V, since K was needed in order to produce the message m_4.

It is important to note the different functions of the messages transmitted in this protocol. The messages $m_1$and m_2 are used to provide secrecy in the transmission of the session key K. On the other hand, m_3 and m_4 are used to provide *key confirmation*, that is, to enable U and V to convince each other that they possess the same session key K. In most key distribution schemes, session) key confirmation can be included as a feature if it is not already present. Usually this is done in a similar fashion as it is done in **Kerberos**, namely by using the new session key K to encrypt known quantities. In **Kerberos**, U uses K to encrypt ID (U) and T, which are already encrypted in m_2. Similarly, V uses K to encrypt $T + 1$.

4.6 DIFFIE-HELLMAN KEY EXCHANGE

In this section, we describe a key predistribution scheme that is a modification of the well-known Diffie-Hellman key exchange protocol that we will discuss a bit later, in Section 8.4. We call this the **Diffie-Hellman Key Predistribution Scheme**. The scheme is computationally secure provided a problem related to the **Discrete Logarithm** problem is intractible.

We will describe the scheme over Z_p , where p is prime, though it can be implemented in any finite group in which the **Discrete Logarithm** problem is intractible. We will assume that a is a primitive element of Z_p, and that the values p and α are publicly known to everyone in the network.

In this scheme, ID(U) will denote certain identification information for each user U in the network, e.g., his or her name, e-mail address, telephone number, or other relevant information. Also, each user U has a secret exponent a_U(where $0 \leq a_U \leq p - 2$), and a corresponding public value

$$b_U = \sim^{a_U} \bmod p.$$

The TA will have a signature scheme with a (public) verification algorithm *ver* TA and a secret signing algorithm *sig* TA. Finally, we will implicitly assume that all information is hashed, using a public hash function, before it is signed. To make the procedures easier to read, we will not include the necessary hashing in the description of the protocols. Certain information pertaining to a user U will be authenticated by means of a *certificate* which is issued and signed by the TA. Each user U will have a certificate

$$C(U) = \left(\mathrm{ID}(\mathrm{U}), b_U, sig_{TA}\left(ID(U), b_U\right)\right).$$

where b_U is formed as described above (note that the TA does not need to know the value of a_U). A certificate for a user U will be issued when U joins the network. Certificates can be stored in a public database, or each user can store his or her own certificate. The signature of the TA on a certificate allows anyone in the network to verify the information it contains. It is very easy for U and V to compute the common key

$$K_{U,V} = \sim^{a_U a_V} \bmod p.$$

We illustrate the algorithm with a small example.

Example 2.2:

Suppose p=25307 and $\sim = 2$ are publicity known (p is prime and ~ is a primitive root modulo p). Suppose U chooses a_U = 3578. Then she computes

$$\begin{aligned} b_U &= \sim^{aU} \bmod p \\ &= 2^{3578} \bmod 25307 \\ &= 6113 \end{aligned}$$

Which is placed on her certificate. Suppose V chooses a_v= 19956. Then he computers

$$\begin{aligned} b_v &= \sim^{av} \bmod p \\ &= 2^{19956} \bmod 25307 \\ &= 7984, \end{aligned}$$

If we do not want to use an on-line key server, then we are forced to use a key agreement protocol to exchange secret keys. The first and best known key agreement protocol is **Diffie-Hellman Key Exchange**. We will assume that p is prime, α is a primitive element of Z_p, and that the values p and α are publicly known. (Alternatively, they could be chosen by U and communicated to V in the first step of the protocol.) **Diffie-**

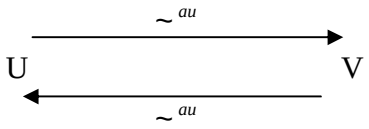
Hellman Key Exchange is presented. At the end of the protocol, U and V have computed the same key

$$K = \sim^{auav} \bmod p$$

This protocol is very similar to **Diffie-Hellman Key Predistribution** described earlier. The difference is that the exponents' a_U and a_V of users U and V (respectively) are chosen anew each time the protocol is run, instead of being fixed. Also, in this protocol, both U and V are assured of key freshness, since the session key depends on both random exponents a_U and a_V.

4.7 THE STATION-TO-STATION PROTOCOL

Diffie-Hellman Key Exchange is supposed to look like this:



Unfortunately, the protocol is vulnerable to an active adversary who uses an *intruder-in-the-middle* attack. There is an episode of *The Lucy Show* in which Vivian Vance is having dinner in a restaurant with a date, and Lucille Ball is hiding under the table. Vivian and her date decide to hold hands under the table. Lucy, trying to avoid detection, holds hands with each of them and they think they are holding hands with each other.

An intruder-in-the-middle attack on the **Diffie-Hellman Key Exchange** protocol works in the same way. W will intercept messages between U and V and substitute his own messages, as indicated in the following diagram:

At the end of the protocol, U has actually established the secret key $\sim^{aua'V}$ with W, and V has established a secret key $\sim^{aua'V}$ with W. When U tries to encrypt a message to send to V, W will be able to decrypt it but V will not. (A similar situation holds if V sends a message to U.)

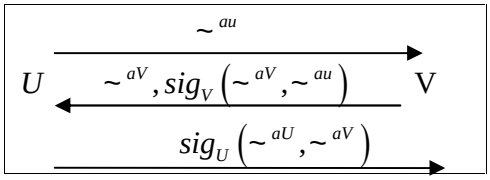
Clearly, it is essential for U and V to make sure that they are exchanging messages with each other and not with W. Before exchanging keys, U and V might carry out a separate protocol to establish each other's identity, for example by using one of the identification schemes. But this offers no protection against an intruder-in-the-middle attack if W simply

remains inactive until after U and V have proved their identities to each other. Hence, the key agreement protocol should itself authenticate the participants' identities at the same time as the key is being established. Such a protocol will be called *authenticated key agreement*.

We will describe an authenticated key agreement protocol which is a modification of **Diffie-Hellman Key Exchange**. The protocol assumes a publicly known prime p and a primitive element α, and it makes use of certificates. Each user U will have a signature scheme with verification algorithm *ver*U and signing algorithm *sig*U. The TA also has a signature scheme with public verification algorithm *ver*TA. Each user U has a certificates

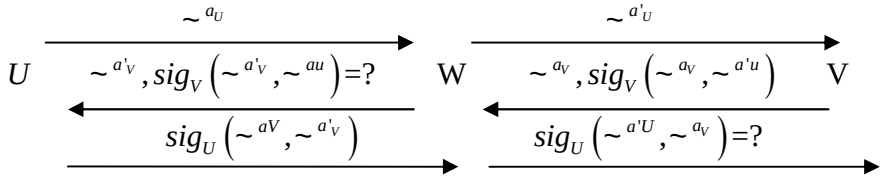
$$C(U) = \left(\mathrm{ID}(\mathrm{U}), ver_U, sig_{TA}\left(ID(U), ver_U\right)\right).$$

where ID(U) is identification information for U. The authenticated key agreement known as the **Station-to-station Protocol** (or **STS** for short) is due to Diffie, Van Oorschot, and Wiener. The protocol is a slight simplification; it can be used in such a way that it is conformant with the ISO 9798-3 protocols. The information exchanged in the simplified **STS protocol** (excluding certificates) is illustrated as follows



Let's see how this protects against an intruder-in-the-middle attack. As before W will intercept $\sim^{aU}$ and replace it with $\sim^{a'U}$. W then receives $\sim^{a'V}, sig_v\left(\sim^{av}, \sim^{a'U}\right)$ from V. He would like to replace $\sim^{a'U}$ it with $\sim^{a'v}$ as before. However this means that he must also replace $sig_v\left(\sim^{av}, \sim^{a'U}\right)$ by $sig_v\left(\sim^{a'v}, \sim^{aU}\right)$ as before. However, this means that he must also replace $sig_V\left(\sim^{a'v}, \sim^{aU}\right)$ since he doesn't know V's signing algorithm sig_V. Similarly ,W is unable to replace $sig_U\left(\sim^{aU}, \sim^{a'V}\right)$ by $sig_V\left(\sim^{a'V}, \sim^{aV}\right)$ because HE does not know U's algorithm.

This is illustrated in the following diagram.



It is the use of signatures that thwarts the intruder in the middle attack.

The protocol as described in 2.6 does not provide key confirmation. How it is easy to modify so that it does by defining

$$yv = ek\left(sigv\left(\sim^{aV}, \sim^{aU}\right)\right)$$

In step 4 and defining

$$y_U = e_K\left(sig_U\left(\sim^{aU}, \sim^{aV}\right)\right)$$

In step 6. (As in Kerberos, we obtain key confirmation by encrypting a known quantity using the new session key). The resulting protocol is known as the Station-to-station Protocol. We leave the

4.8 MTI KEY AGREEMENT PROTOCOLS

Matsumoto, Takashima, and Imai have constructed several interesting key agreement protocols by modifying **Diffie-Hellman Key Exchange**. These protocols, which we call **MTI** protocols, do not require that U and V compute any signatures. They are *two-pass protocols* since there are only two separate transmissions of information performed (one from U to V and one from V to U). In contrast, the **STS** protocol is a three-pass protocol.

We present one of the **MTI** protocols. The setting for this protocol is the same as for **Diffie-Hellman Key Predistribution**. We assume a publicly known prime p and a primitive element α. Each user U has an ID string, ID(U), a secret exponent aU($0 \le a\text{U} \le p - 2$), and a corresponding public value

$$b_U = \sim^{a_U} \bmod p.$$

The TA has a signature scheme with a (public) verification algorithm *ver*TA and a secret signing algorithm *sig*TA. Each user U will have a certificate

$$C(U) = \left(\text{ID}(\text{U}), b_U, sig_{TA}\left(\text{ID}(\text{U}), \text{b}_\text{U}\right)\right),$$

where b_U is formed as described above

We present the MTI key agreement protocol in Figure 2.7. At the end of the protocol U and V have both computed the same key.

$$K = \sim^{ruav+rvau} \bmod p.$$

We give an example to illustrate this protocol

Suppose p=27803 and are publicity known. Assume U chooses the she will compute

$$b_U = 5^{21131} \bmod 27803 = 21420.$$

Which is placed on her certificate. As well assume V chooses a_V=17555. Then he will compute

$$b_V = 5^{17555} \bmod 27803 = 17100.$$

which is placed on his certificate

Now suppose that Uchooses r_U=169; then she will send the value

$$s_U = 5^{169} \bmod 27803 = 6268.$$

To V. Suppose that U chooses r_U =169; then she will send the value

$$s_V = 5^{23456} \bmod 27803 = 26759.$$

to U

Now U can complete the key

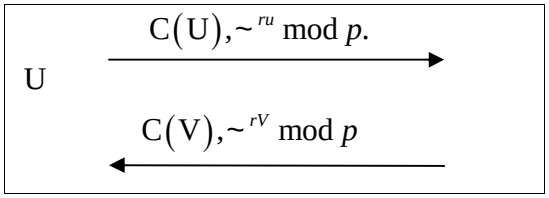
$$\begin{aligned} K_{UV} &= s_V{}^{GU} by^{ru} \bmod p \\ &= 26759^{21131} 17100^{169} \bmod 27803 \\ &= 21600 \end{aligned}$$

And V can compute the key

$$\begin{aligned} K_{U,V} &= s_U{}^{av} b_U{}^{ru} \bmod p \\ &= 6268^{17555} 21420^{23456} \bmod 27803 \\ &= 21600, \end{aligned}$$

Thus U and V have computed the same key

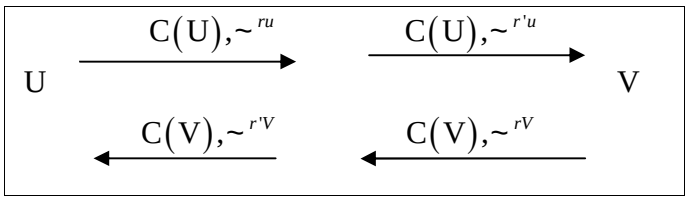
The information transmitted during the protocol is depicted as follows:



Let's look at the security of the scheme. It is not too difficult to show that the security of the **MTI** protocol against a passive adversary is

exactly the same as the **Diffie-Hellman** problem — see the exercises. As with many protocols, proving security in the presence of an active adversary is problematic. We will not attempt to prove anything in this regard, and we limit ourselves to some informal arguments.

Here is one threat we might consider: Without the use of signatures during the protocol, it might appear that there is no protection against an intruder-in-the-middle attack. Indeed, it is possible that W might alter the values that U and V send each other. We depict one typical scenario that might arise, as follows:



In this situation, U and V will compute different keys: U will compute while V will compute

$$K = \sim^{ruav+r'vau} \bmod p$$
$$K = \sim^{r'uav+r'vau} \bmod p$$

However, neither of the key computations of U or V can be carried out by W, since they require knowledge of the secret exponents *a*U and *a*V , respectively. So even though U and V have computed different keys (which will of course be useless to them), neither of these keys can be computed by W (assuming the intractibility of the **Discrete Log** problem). In other words, both U and V are assured that the other is the only user in the network that could compute the key that they have computed. This property is sometimes called *implicit key authentication*.

4.9 KEY AGREEMENT USING SELF-CERTIFYING KEYS

In this section, we describe a method of key agreement, due to Girault, that does not require certificates. The value of a public key and the identity or its owner implicitly authenticate each other.

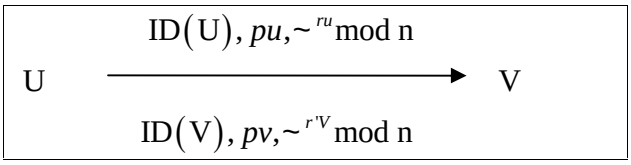
The Girault scheme combines features of RSA and discrete logarithms. Suppose n=pq, where $p=2p_1+1$, $q=2q_1+1$, and p, q, p_1 and q1 are all large primes. The multiplicative group $\mathbb{Z}n^*$ isomorphic to $\mathbb{Z}p^* \times \mathbb{Z}q^*$. The maximum order of any element in $\mathbb{Z}n^*$ is therefore the least common multiple of p-1 and q-1 or $2p_1q_1$. Let ~ be an element of order $2p_1q_1$. Then the cyclic subgroup of $\mathbb{Z}n^*$generated by ~ is a suitable setting for Discrete Logarithm problem.

In Girault scheme, the factorization of n is known only to the TA. The values n and ~ are public, but p, q, p_1 and q1 are all secret. The TAchooses a public RSA encryption exponent, which we will denote by e. The corresponding decryption exponent, d, is secret (recall that $d=e^{-1}$ mod ¤ (n))

Each user U has an ID string ID(U), as in previous schemes. A user U obtains a *self-certifying public key*, P_U , from the TA as indicated in Figure 2.. Observe that U needs the help of the TA to produce p_U. Note also that

$$b_U = p_U{}^e + \mathrm{ID(U)} \bmod n$$

can be computed from p_U and ID(U) using publicly available information. The **Girault Key Agreement Protocol** is presented. The information transmitted during the protocol is depicted as follows:



At the end of protocol, U and V each have computed the key
$K = \sim^{ruav+rvau} \bmod n$

Here is an example of key exchange using the Girault Scheme.

Example 2.4:

Suppose p=839 and q=863. Then n=724057 and ¤(n)=722356. The element ~ =5 has order $2p_1q_1$= ¤(n)/2. Suppose the TA chooses d=125777 as the RSA decryption exponent; then e = 84453

Suppose U has ID(U)=500021 and a_U=111899. Then b_U=488889 and p_U= 650704. Suppose also that V has ID(V) = 50022 and a_V = 123456. Then b_V = 111692 and p_V= 683556.

Now, Uand Vwant to exchange a key. Suppose U chooses r_U=56381, which means that s_U= 171007. Further, suppose V chooses r_V=356935, which means that s_V=320688.

Then both U and V will compute the same key K=42869.

Let's consider how the self-certifying keys guard against one specific type of attack. Since the values b_U, p_U, and ID(U) are not signed by the TA, there is no way for anyone else to verify their authenticity directly.Suppose this information is forged by W(i.e. it is not produced in cooperation with the TA), who wants to masquerade is forged by W (i.e.,

it is not produced in cooperation with the TA), who wants to masquerade as U. If W starts with ID(U) and fake value b`$_U$, then there is no way for her to compute the exponent a`$_U$ corresponding b`$_U$, if the Discrete Log problem is intractable. Without a`$_U$, computation cannot be performed by W(who is pretending to be U).

Girault Key Agreement Protocol The situation is similar if W acts as an intruder-in-the-middle. W will be able to prevent U and V from computing a common key, but W is unable to duplicate the computations of either U or V. Thus the scheme provides implicit key authentication, as did the **MTI** protocol.

An attentive reader might wonder why U is required to supply the value *a*U to the TA. Indeed, the TA can compute *p*U directly from *b*U, without knowing *a*U. Actually, the important thing here is that the TA should be convinced that U knows the value of *a*U before the TA computes *p*U for U.

We illustrate this point by showing how the scheme can be attacked if the TA indiscriminately issues public keys *p*U to users without first checking that they possess the value *a*U corresponding to their *b*U.

Suppose W chooses a fake value a`$_U$, and computes the corresponding value

$$b'_U = \sim^{a'U} \bmod n$$

Here is how he can determine the corresponding public key

$$p'_V = \left(b'_U - \mathrm{ID(U)}\right)^d \bmod n$$

We will compute

$$b'_W = b'_U - \mathrm{ID(U)}+\mathrm{ID(W)}$$

and then given b`$_W$ and ID(W) to the TA. Suppose the TA issues the public key

$$p'_W = (b'_W - \mathrm{ID(W)})^d \bmod \mathrm{n}$$

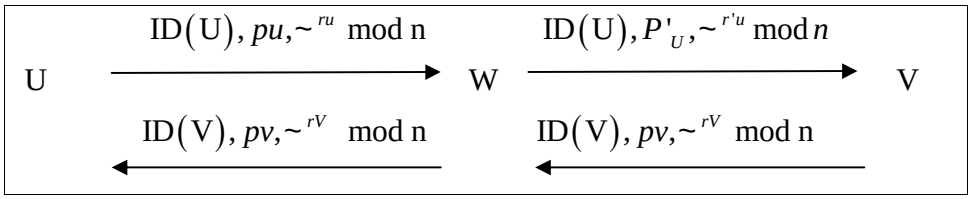
To W using the fact that

$$(b'_W - \mathrm{ID(W)}) \equiv b'_U - \mathrm{ID(U)}) \bmod \mathrm{n}$$

it is immediate that

$$p'_W = p'_U$$

Now at some later time suppose U and V execute the protocol, and W substitute information as follows.



Whereas V will compute the key

$$K'_W = \sim^{r\,'\mathrm{Uav}+rva'U}$$

Now U will compute the key

$$K'_W = \sim^{r\,\mathrm{Uav}+\mathrm{rvaU}} \bmod n$$

W can compute a as

$$K' = sv^{a'U}\left(pv^e + ID(V)\right)^{r'U} m \bmod n$$

Thus Wand Vshare a key, but V thinks he is sharing a key with U. So W will be able to descrypt message sent by V to U

4.10 LET US SUM UP

Thus, we have studied the basic concepts about key predistribution. The Diffie-Hellman key exchange algorithm and Blom's scheme has been described briefly. The MIT key agreement and station to station protocol has been explained.

4.11 LIST OF REFERENCES

- **Cryptography: Theory and Practice,** *Douglas Stinson,* CRC Press, CRC Press LLC
- **Cryptography and Network Security Principles and Practices**, Fourth Edition, William Stallings, PHI(Pearson)

4.12 UNIT END EXERCISES

1. What is key predistribution? Explain the concept.
2. Explain the Diffie-Hellman key exchange algorithm
3. Describe Station-to Station Protocol.
4. Write a short note on Blom's scheme.
5. Explain MTI key agreement protocol.

*****

UNIT III

5

NETWORK SECURITY

Unit Structure

5.0 INTRODUCTION

In 1994, the Internet Architecture Board (IAB) issued a report entitled "Security in the Internet Architecture" (RFC 1636). This chapter is going to focus on security threats and issues.

5.1 SECURITY TRENDS

The report stated the general consensus that the Internet needs more and better security, and it identified key areas for security mechanisms. Among these were the need to secure the network infrastructure from unauthorized monitoring and control of network traffic and the need to secure end-user-to-end-user traffic using authentication and encryption mechanisms.

These concerns are fully justified. As confirmation, consider the trends reported by the Computer Emergency Response Team (CERT) Coordination Center (CERT/CC). Figure 1.1a shows the trend in Internet-related vulnerabilities reported to CERT over a 10-year period. These include security weaknesses in the operating systems of attached computers (e.g., Windows, Linux) as well as vulnerabilities in Internet routers and other network devices. Figure 1.1b shows the number of security related incidents reported to CERT. These include denial of service attacks; IP spoofing, in which intruders create packets with false IP addresses and exploit applications that use authentication based on IP; and

various forms of eavesdropping and packet sniffing, in which attackers read transmitted information, including logon information and database contents.

5.2 THE OSI SECURITY ARCHITECTURE

To assess effectively the security needs of an organization and to evaluate and choose various security products and policies, the manager responsible for security needs some systematic way of defining the requirements for security and characterizing the approaches to satisfying those requirements. This is difficult enough in a centralized data processing environment; with the use of local and wide area networks, the problems are compounded.

ITU-T Recommendation X.800, Security Architecture for OSI, defines such a systematic approach. The OSI security architecture is useful to managers as a way of organizing the task of providing security. Furthermore, because this architecture was developed as an international standard, computer and communications vendors have developed security features for their products and services that relate to this structured definition of services and mechanisms

For our purposes, the OSI security architecture provides a useful, if abstract, overview of many of the concepts that this book deals with. The OSI security architecture focuses on security attacks, mechanisms, and services. These can be defined briefly as follows:

- **Security attack:** Any action that compromises the security of information owned by an organization.
- **Security mechanism:** A process (or a device incorporating such a process) that is designed to detect, prevent, or recover from a security attack.
- **Security service:** A processing or communication 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.

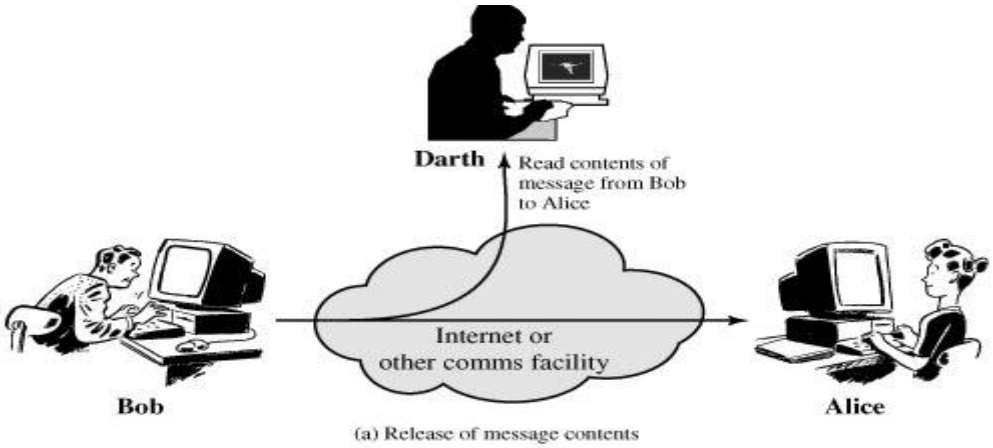
5.3 SECURITY ATTACKS

A useful means of classifying security attacks, used both in X.800 and RFC 2828, is in terms of passive attacks and active attacks. A passive attack attempts to learn or make use of information from the system but does not affect system resources. An active attack attempts to alter system resources or affect their operation.

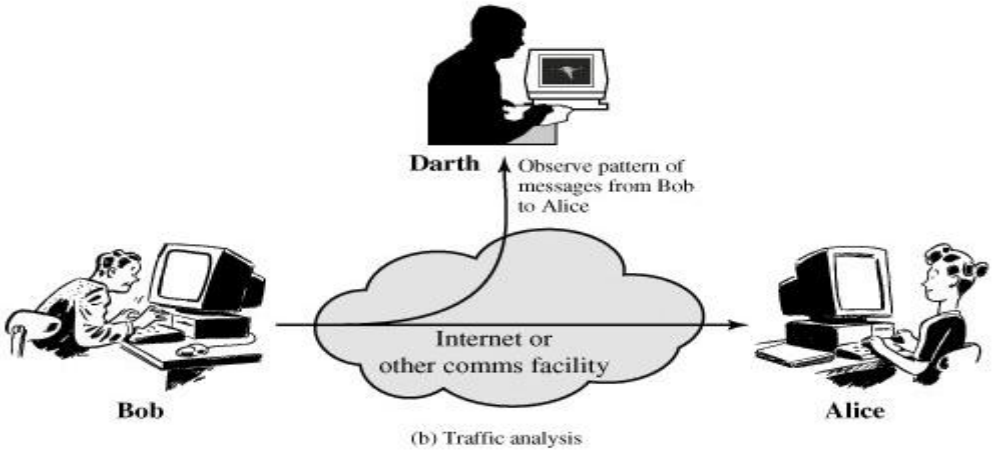
Passive Attacks:

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. Two types of passive attacks are release of message contents and traffic analysis.

The release of message contents is easily understood (Figure 1.3a). A telephone conversation, an electronic mail message, and a transferred file may contain sensitive or confidential information. We would like to prevent an opponent from learning the contents of these transmissions.



(a) Release of message contents



(b) Traffic analysis

A second type of passive attack, traffic analysis, is subtler (Figure 1.3b). Suppose that we had a way of masking the contents of messages or other information traffic so that opponents, even if they captured the message, could not extract the information from the message. The common technique for masking contents is encryption. If we had encryption protection in place, an opponent might still be able to observe the pattern of these messages. The opponent could determine the location and identity of communicating hosts and could observe the frequency and length of messages being exchanged. This information might be useful in guessing the nature of the communication that was taking place.

Passive attacks are very difficult to detect because they do not involve any alteration of the data. Typically, the message traffic is sent and received in an apparently normal fashion and neither the sender nor receiver is aware that a third party has read the messages or observed the traffic pattern.

However, it is feasible to prevent the success of these attacks, usually by means of encryption. Thus,

Active Attacks:

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

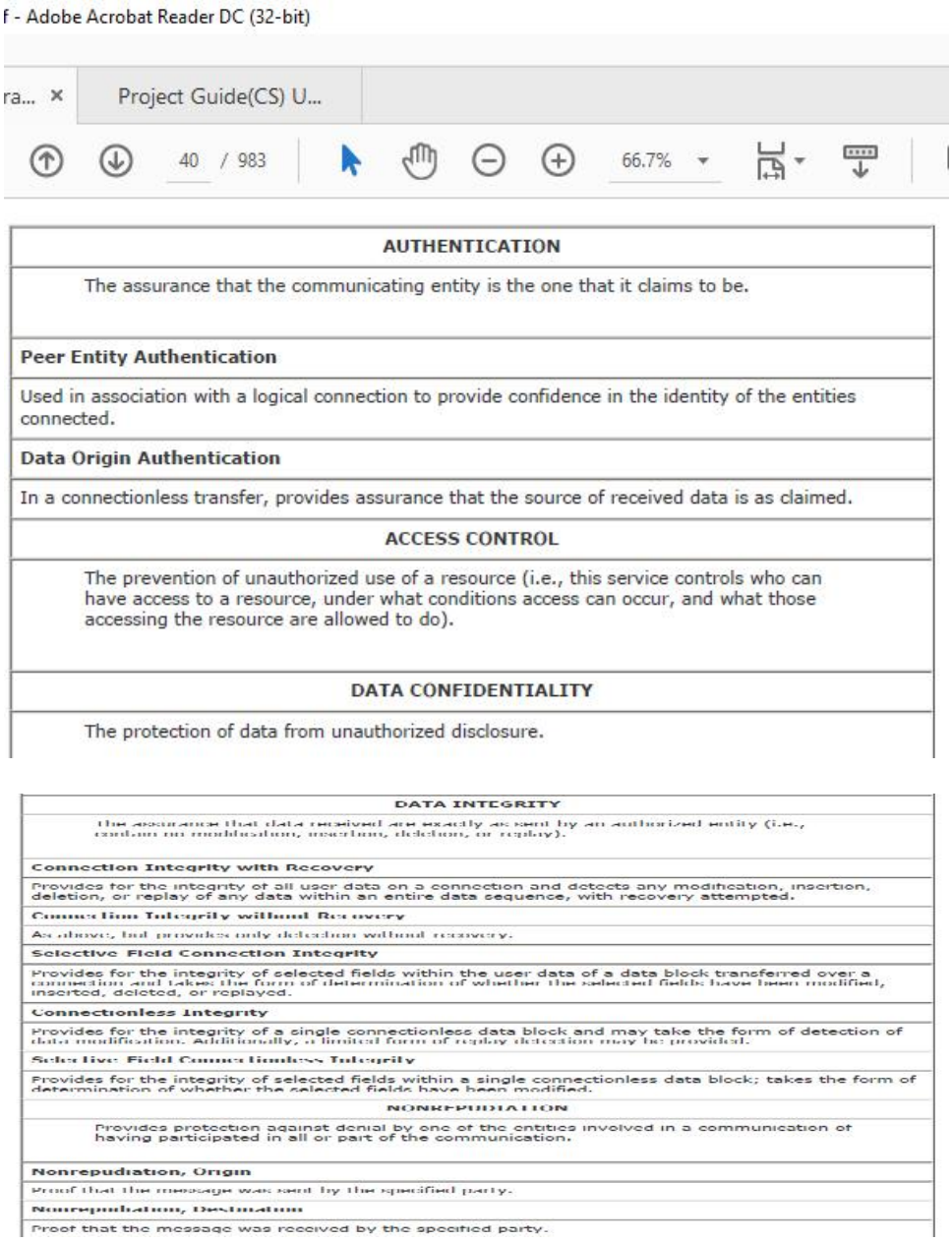
A masquerade takes place when one entity pretends to be a different entity (Figure 1.4a). A masquerade attack usually includes one of the other forms of active attack. For example, authentication sequences can be captured and replayed after a valid authentication sequence has taken place, thus enabling an authorized entity with few privileges to obtain extra privileges by impersonating an entity that has those privileges.

Modification of messages simply means that some portion of a legitimate message is altered, or that messages are delayed or reordered, to produce an unauthorized effect (Figure 1.4c). For example, a message meaning "Allow John Smith to read confidential file accounts" is modified to mean "Allow Fred Brown to read confidential file accounts."

The denial of service prevents or inhibits the normal use or management of communications facilities (Figure 1.4d). This attack may have a specific target; for example, an entity may suppress all messages directed to a particular destination (e.g., the security audit service). Another form of service denial is the disruption of an entire network, either by disabling the network or by overloading it with messages so as to degrade performance.

5.4 SECURITY SERVICES

X.800 defines a security service as a service provided by a protocol layer of communicating open systems, which ensures adequate security of the systems or of data transfers. Perhaps a clearer definition is found in RFC 2828, which provides the following definition: a processing or communication service that is provided by a system to give a specific kind of protection to system resources; security services implement security policies and are implemented by security mechanisms.



Authentication:

The authentication service is concerned with assuring that a communication is authentic. In the case of a single message, such as a warning or alarm signal, the function of the authentication service is to assure the recipient that the message is from the source that it claims to be from. In the case of an ongoing interaction, such as the connection of a terminal to a host, two aspects are involved. First, at the time of connection initiation, the service assures that the two entities are authentic, that is, that each is the entity that it claims to be. Second, the service must assure that the connection is not interfered with in such a way that a third party can masquerade as one of the two legitimate parties for the purposes of unauthorized transmission or reception.

Two specific authentication services are defined in X.800:

- **Peer entity authentication:** Provides for the corroboration of the identity of a peer entity in an association. It is provided for use at the establishment of, or at times during the data transfer phase of, a connection. It attempts to provide confidence that an entity is not performing either a masquerade or an unauthorized replay of a previous connection.
- **Data origin authentication:** Provides for the corroboration of the source of a data unit. It does not provide protection against the duplication or modification of data units. This type of service supports applications like electronic mail where there are no prior interactions between the communicating entities.

Access Control:

In the context of network security, access control is the ability to limit and control the access to host systems and applications via communications links. To achieve this, each entity trying to gain access must first be identified, or authenticated, so that access rights can be tailored to the individual.

Data Confidentiality:

Confidentiality is the protection of transmitted data from passive attacks. With respect to the content of a data transmission, several levels of protection can be identified. The broadest service protects all user data transmitted between two users over a period of time. For example, when a TCP connection is set up between two systems, this broad protection prevents the release of any user data transmitted over the TCP connection. Narrower forms of this service can also be defined, including the protection of a single message or even specific fields within a message. These refinements are less useful than the broad approach and may even be more complex and expensive to implement.

The other aspect of confidentiality is the protection of traffic flow from analysis. This requires that an attacker not be able to observe the source and destination, frequency, length, or other characteristics of the traffic on a communications facility.

Data Integrity:

As with confidentiality, integrity can apply to a stream of messages, a single message, or selected fields within a message. Again, the most useful and straightforward approach is total stream protection.

A connection-oriented integrity service, one that deals with a stream of messages, assures that messages are received as sent, with no

duplication, insertion, modification, reordering, or replays. The destruction of data is also covered under this service. Thus, the connection-oriented integrity service addresses both message stream modification and denial of service. On the other hand, a connectionless integrity service, one that deals with individual messages without regard to any larger context, generally provides protection against message modification only.

We can make a distinction between the service with and without recovery. Because the integrity service relates to active attacks, we are concerned with detection rather than prevention. If a violation of integrity is detected, then the service may simply report this violation, and some other portion of software or human intervention is required to recover from the violation. Alternatively, there are mechanisms available to recover from the loss of integrity of data, as we will review subsequently. The incorporation of automated recovery mechanisms is, in general, the more attractive alternative.

Nonrepudiation:

Nonrepudiation prevents either sender or receiver from denying a transmitted message. Thus, when a message is sent, the receiver can prove that the alleged sender in fact sent the message. Similarly, when a message is received, the sender can prove that the alleged receiver in fact received the message.

Availability Service:

Both X.800 and RFC 2828 define availability to be the property of a system or a system resource being accessible and usable upon demand by an authorized system entity, according to performance specifications for the system (i.e., a system is available if it provides services according to the system design whenever users request them). A variety of attacks can result in the loss of or reduction in availability. Some of these attacks are amenable to automated countermeasures, such as authentication and encryption, whereas others require some sort of physical action to prevent or recover from loss of availability of elements of a distributed system.
X.800 treats availability as a property to be associated with various security services. However, it makes sense to call out specifically an availability service. An availability service is one that protects a system to ensure its availability. This service addresses the security concerns raised by denial-of-service attacks. It depends on proper management and control of system resources and thus depends on access control service and other security services.

5.5 SECURITY MECHANISMS

Table 1.3 lists the security mechanisms defined in X.800. As can be seen the mechanisms are divided into those that are implemented in a

specific protocol layer and those that are not specific to any particular protocol layer or security service.

These mechanisms will be covered in the appropriate places in the book and so we do not elaborate now, except to comment on the definition of encipherment. X.800 distinguishes between reversible encipherment mechanisms and irreversible encipherment mechanisms. A reversible encipherment mechanism is simply an encryption algorithm that allows data to be encrypted and subsequently decrypted. Irreversible encipherment mechanisms include hash algorithms and message authentication codes, which are used in digital signature and message authentication applications.

SPECIFIC SECURITY MECHANISMS
May be incorporated into the appropriate protocol layer in order to provide some of the OSI security services.
Encipherment
The use of mathematical algorithms to transform data into a form that is not readily intelligible. The transformation and subsequent recovery of the data depend on an algorithm and zero or more encryption keys.
Digital Signature
Data appended to, or a cryptographic transformation of, a data unit that allows a recipient of the data unit to prove the source and integrity of the data unit and protect against forgery (e.g., by the recipient).
Access Control
A variety of mechanisms that enforce access rights to resources.
Data Integrity
A variety of mechanisms used to assure the integrity of a data unit or stream of data units.
Authentication Exchange
A mechanism intended to ensure the identity of an entity by means of information exchange.
Traffic Padding
The insertion of bits into gaps in a data stream to frustrate traffic analysis attempts.
Routing Control
Enables selection of particular physically secure routes for certain data and allows routing changes, especially when a breach of security is suspected.
Notarization
The use of a trusted third party to assure certain properties of a data exchange.
PERVASIVE SECURITY MECHANISMS
Mechanisms that are not specific to any particular OSI security service or protocol layer.
Trusted Functionality
That which is perceived to be correct with respect to some criteria (e.g., as established by a security policy).

Connection Confidentiality
The protection of all user data on a connection.
Connectionless Confidentiality
The protection of all user data in a single data block
Selective-Field Confidentiality
The confidentiality of selected fields within the user data on a connection or in a single data block.
Traffic Flow Confidentiality
The protection of the information that might be derived from observation of traffic flows.

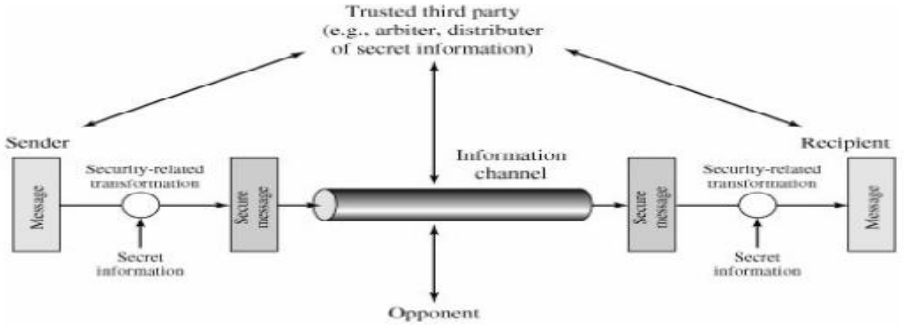
Security Label
The marking bound to a resource (which may be a data unit) that names or designates the security attributes of that resource.
Event Detection
Detection of security-relevant events.
Security Audit Trail
Data collected and potentially used to facilitate a security audit, which is an independent review and examination of system records and activities.
Security Recovery
Deals with requests from mechanisms, such as event handling and management functions, and takes recovery actions.

	Mechanism							
Service	Encipherment	Digital Signature	Access Control	Data Integrity	Authentication Exchange	Traffic Padding	Routing Control	Notarization
Peer entity authentication	Y	Y			Y			
Data origin authentication	Y	Y						
Access control			Y					
Confidentiality	Y						Y	
Traffic flow confidentiality	Y					Y	Y	
Data integrity	Y	Y		Y				
Nonrepudiation		Y		Y				Y
Availability				Y	Y			

,

5.6A MODEL FOR NETWORK SECURITY

A model for much of what we will be discussing is captured, in very general terms, in Figure 1.5. 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.



Security aspects come into play when it is necessary or desirable to protect the information transmission from an opponent who may present a threat to confidentiality, authenticity, and so on. All the techniques for providing security have two components:

- A security-related transformation on the information to be sent. Examples include the encryption of the message, which scrambles the message so that it is unreadable by the opponent, and the addition of a code based on the contents of the message, which can be used to verify the identity of the sender
- Some secret information shared by the two principals and, it is hoped, unknown to the opponent. An example is an encryption key used in conjunction with the transformation to scramble the message before transmission and unscramble it on reception.

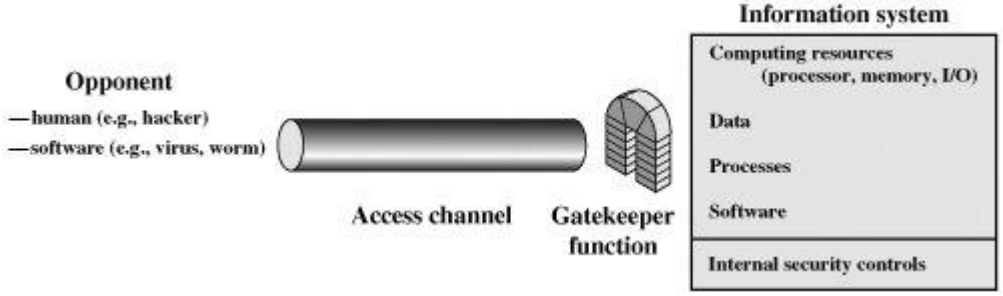
A trusted third party may be needed to achieve secure transmission. For example, a third party may be responsible for distributing the secret information to the two principals while keeping it

from any opponent. Or a third party may be needed to arbitrate disputes between the two principals concerning the authenticity of a message transmission.

This general model shows that there are four basic tasks in designing a particular security service:

- Design an algorithm for performing the security-related transformation. The algorithm should besuch that an opponent cannot defeat its purpose.
- Generate the secret information to be used with the algorithm.
- Develop methods for the distribution and sharing of the secret information.
- Specify a protocol to be used by the two principals that makes use of the security algorithm and the secret information to achieve a particular security service.

Parts One through Three of this book concentrates on the types of security mechanisms and services that fit into the model shown in Figure 1.5. However, there are other security-related situations of interest that do not neatly fit this model but that are considered in this book. A general model of these other situations is illustrated by Figure 1.6, which reflects a concern for protecting an information system from unwanted access. Most readers are familiar with the concerns caused by the existence of hackers, who attempt to penetrate systems that can be accessed over a network. The hacker can be someone who, with no malign intent, simply gets satisfaction from breaking and entering a computer system. Or, the intruder can be a disgruntled employee who wishes to do damage, or a criminal who seeks to exploit computer assets for financial gain (e.g., obtaining credit card numbers or performing illegal money transfers).



Another type of unwanted access is the placement in a computer system of logic that exploits vulnerabilities in the system and that can affect application programs as well as utility programs, such as editors and compilers. Programs can present two kinds of threats:

- Information access threats intercept or modify data on behalf of users who should not have access to that data.
- Service threats exploit service flaws in computers to inhibit use by legitimate users.

Viruses and worms are two examples of software attacks. Such attacks can be introduced into a system by means of a disk that contains the unwanted logic concealed in otherwise useful software. They can also be inserted into a system across a network; this latter mechanism is of more concern in network security.

The security mechanisms needed to cope with unwanted access fall into two broad categories (see Figure 1.6). The first category might be termed a gatekeeper function. It includes password-based login procedures that are designed to deny access to all but authorized users and screening logic that is designed to detect and reject worms, viruses, and other similar attacks. Once either an unwanted user or unwanted software gains access, the second line of defense consists of a variety of internal controls that monitor activity and analyze stored information in an attempt to detect the presence of unwantedintruders.

5.7 SUMMARY

This chapter is mainly focusing on security threads. We have tried to cover all main aspects regarding to security.

5.8 REFERENCES AND BIBLIOGRAPHY: -

- Cryptography: Theory and Practice, Douglas Stinson, CRC Press, CRC Press LLC
- Cryptography and Network Security Principles and Practices, Fourth Edition, William Stallings, PHI(Pearson),

5.9 EXERCISE

1. What is the OSI security architecture?
2. What is the difference between passive and active security threats?
3. List and briefly define categories of passive and active security attacks.
4. List and briefly define categories of security services.
5. List and briefly define categories of security mechanisms.

*****

UNIT IV: NETWORKING

6

AUTHENTICATION APPLICATIONS

Unit structure

6.0 INTRODUCTION

Authentication Applications: Kerberos, X.509 Authentication Service, Public-Key Infrastructure, Recommended Reading and Web Sites, Key Terms, Review Questions, and Problems, A Kerberos Encryption Techniques, Electronic Mail Security, Pretty Good Privacy, S/MIME, Key Terms, Review Questions, and Problems, A Data Compression Using Zip, Radix-64 Conversion, PGP Random Number Generation

Kerberos is an authentication service designed for use in a distributed environment.

- Kerberos makes use of a trusted third-part authentication service that enables clients and servers to establish authenticated communication.
- X.509 defines the format for public-key certificates. This format is widely used in a variety of applications.
- A public key infrastructure (PKI) is defined as the set of hardware, software, people, policies, and procedures needed to create, manage, store, distribute, and revoke digital certificates based on asymmetric cryptography.

- Typically, PKI implementations make use of X.509 certificates

6.1 KERBEROS

Kerberos[1] is an authentication service developed as part of Project Athena at MIT. The problem that Kerberos addresses is this: Assume an open distributed environment in which users at workstations wish to access services on servers distributed throughout the network. We would like for servers to be able to restrict access to authorized users and to be able to authenticate requests for service. In this environment, a workstation cannot be trusted to identify its users correctly to network services. In particular, the following three threats exist

- A user may gain access to a particular workstation and pretend to be another user operating from that workstation.
- A user may alter the network address of a workstation so that the requests sent from the altered workstation appear to come from the impersonated workstation.
- A user may eavesdrop on exchanges and use a replay attack to gain entrance to a server or to disrupt operations.

In any of these cases, an unauthorized user may be able to gain access to services and data that he or she is not authorized to access. Rather than building in elaborate authentication protocols at each server, Kerberos provides a centralized authentication server whose function is to authenticate users to servers and servers to users.

6.2 X.509 AUTHENTICATION SERVICE

ITU-T recommendation X.509 is part of the X.500 series of recommendations that define a directory service. The directory is, in effect, a server or distributed set of servers that maintains a database of information about users. The information includes a mapping from user name to network address, as well as other attributes and information about the users.

X.509 defines a framework for the provision of authentication services by the X.500 directory to its users. 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 (Chapter 15), IP Security (Chapter 16), and SSL/TLS and SET .

X.509 was initially issued in 1988. The standard was subsequently revised to address some of the security concerns documented in [IANS90] and [MITC90]; a revised recommendation was issued in 1993. A third version was issued in 1995 and revised in 2000.

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. Again, the standard does not dictate a specific hash algorithm. The 1988 recommendation included the description of a recommended hash algorithm; this algorithm has since been shown to be insecure and was dropped from the 1993 recommendation.

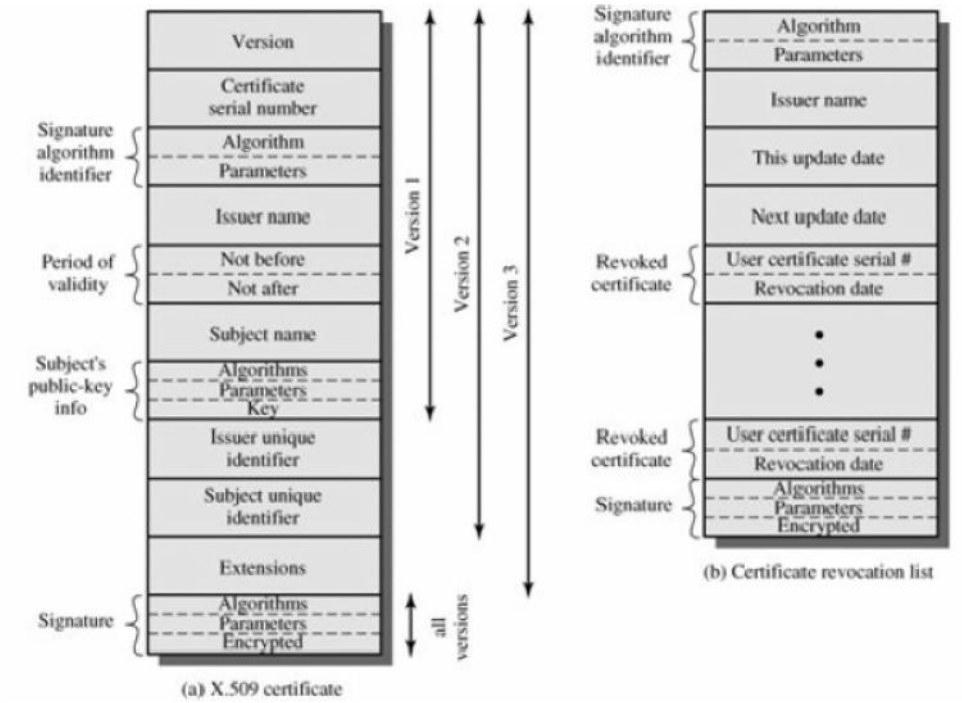
Certificates:

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. The directory server itself is not responsible for the creation of public keys or for the certification function; it merely provides an easily accessible location for users to obtain certificates.

Figure 14.4a shows the general format of a certificate, which includes the following elements:

- **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 identifier.



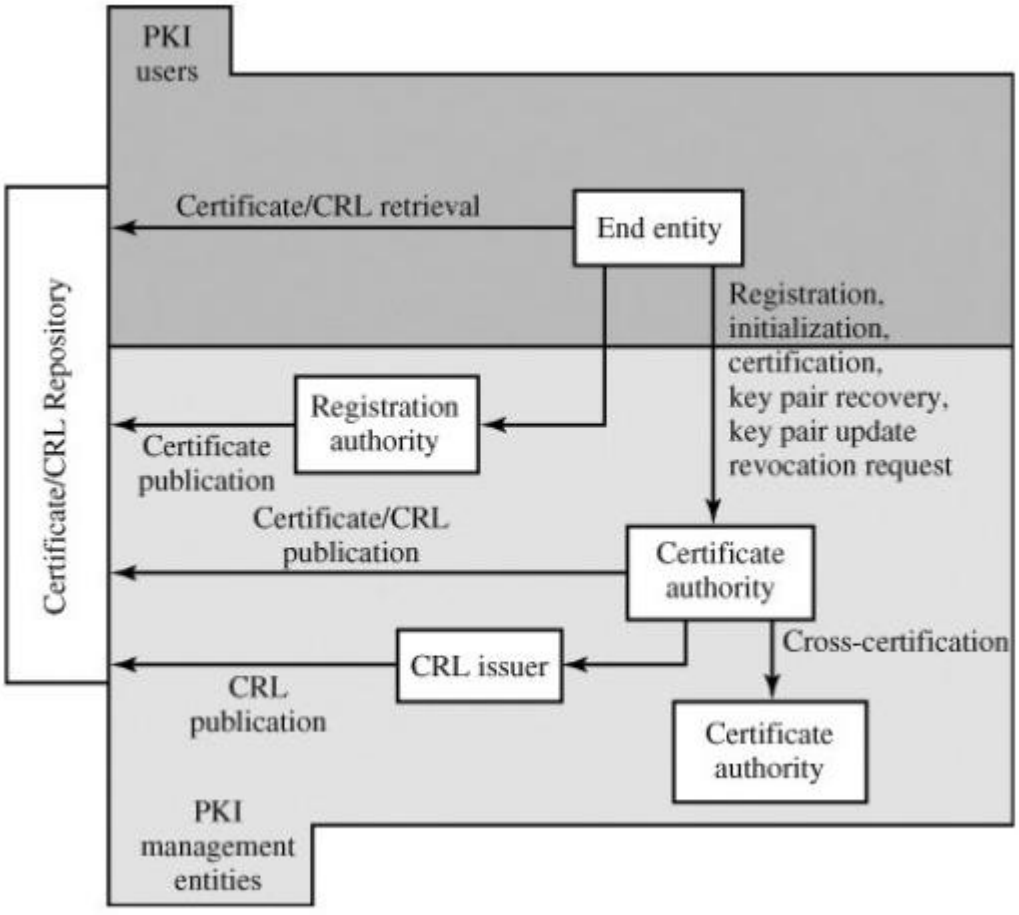
6.3 PUBLIC-KEY INFRASTRUCTURE

RFC 2822 (Internet Security Glossary) defines public-key infrastructure (PKI) as the set of hardware, software, people, policies, and procedures needed to create, manage, store, distribute, and revoke digital certificates based on asymmetric cryptography. The principal objective for developing a PKI is to enable secure, convenient, and efficient acquisition of public keys. The Internet Engineering Task Force

(IETF) Public Key Infrastructure X.509 (PKIX) working group has been the driving force behind setting up a formal (and generic) model based on X.509 that is suitable for deploying a certificate-based architecture on the Internet. This section describes the PKIX model.

Figure 14.7 shows the interrelationship among the key elements of the PKIX model. These elements are

- **End entity:** A generic term used to denote end users, devices (e.g., servers, routers), or any other entity that can be identified in the subject field of a public key certificate. End entities typically consume and/or support PKI-related services.
- **Certification authority (CA):** The issuer of certificates and (usually) certificate revocation lists (CRLs). It may also support a variety of administrative functions, although these are often delegated to one or more Registration Authorities.
- **Registration authority (RA):** An optional component that can assume a number of administrative functions from the CA. The RA is often associated with the End Entity registration process, but can assist in a number of other areas as well.
- **CRL issuer:** An optional component that a CA can delegate to publish CRLs.
- **Repository:** A generic term used to denote any method for storing certificates and CRLs so that they can be retrieved by End Entities.



PKIX Management Functions

PKIX identifies a number of management functions that potentially need to be supported by management protocols. These are indicated in Figure 6.7 and include the following:

- **Registration:** This is the process whereby a user first makes itself known to a CA (directly, or through an RA), prior to that CA issuing a certificate or certificates for that user. Registration begins the process of enrolling in a PKI. Registration usually involves some offline or online procedure for mutual authentication. Typically, the end entity is issued one or more shared secret keys used for subsequent authentication.

Initialization: Before a client system can operate securely, it is necessary to install key materials that have the appropriate relationship with keys stored elsewhere in the infrastructure.

For example, the client needs to be securely initialized with the public key and other assured information of the trusted CA(s), to be used in validating certificate paths.

- **Certification:** This is the process in which a CA issues a certificate for a user's public key, and returns that certificate to the user's client system and/or posts that certificate in a repository.
- **Key pair recovery:** Key pairs can be used to support digital signature creation and verification, encryption and decryption, or both. When a key pair is used for encryption/decryption, it is important to provide a mechanism to recover the necessary decryption keys when normal access to the keying material is no longer possible, otherwise it will not be possible to recover the encrypted data. Loss of access to the decryption key can result from forgotten passwords/PINs, corrupted disk drives, damage to hardware tokens, and so on. Key pair recovery allows end entities to restore their encryption/decryption key pair from an authorized key backup facility (typically, the CA that issued the End Entity's certificate).
- **Key pair update:** All key pairs need to be updated regularly (i.e., replaced with a new key pair) and new certificates issued. Update is required when the certificate lifetime expires and as a result of certificate revocation.
- **Revocation request:** An authorized person advises a CA of an abnormal situation requiring certificate revocation. Reasons for revocation include private key compromise, change in affiliation, and name change.
- **Cross certification:** Two CAs exchange information used in establishing a cross-certificate. A cross-certificate is a certificate issued by one CA to another CA that contains a CA signature key used for issuing certificates.

6.4 THE KERBOSE ENCRYPTION TECHNIQUES

Kerberos can use a variety of cipher algorithms to protect data. A Kerberos encryption type (also known as an enctype) is a specific combination of a cipher algorithm with an integrity algorithm to provide both confidentiality and integrity to data.

Enctypes in requests:

Clients make two types of requests (KDC-REQ) to the KDC: AS-REQs and TGS-REQs. The client uses the AS-REQ to obtain initial tickets (typically a Ticket-Granting Ticket (TGT)), and uses the TGS-REQ to obtain service tickets.

The KDC uses three different keys when issuing a ticket to a client:

The long-term key of the service: the KDC uses this to encrypt the actual service ticket. The KDC only uses the first long-term key in the most recent kvno for this purpose.

The session key: the KDC randomly chooses this key and places one copy inside the ticket and the other copy inside the encrypted part of the reply.

The reply-encrypting key: the KDC uses this to encrypt the reply it sends to the client. For AS replies, this is a long-term key of the client principal. For TGS replies, this is either the session key of the authenticating ticket, or a subsession key.

Each of these keys is of a specific enctype.

Each request type allows the client to submit a list of enctypes that it is willing to accept. For the AS-REQ, this list affects both the session key selection and the reply-encrypting key selection. For the TGS-REQ, this list only affects the session key selection.

Session key selection:

The KDC chooses the session key enctype by taking the intersection of its permitted_enctypes list, the list of long-term keys for the most recent kvno of the service, and the client's requested list of enctypes.

Starting in krb5-1.11, it is possible to set a string attribute on a service principal to control what session key enctypes the KDC may issue for service tickets for that principal. See set_string in kadmin for details.

Choosing enctypes for a service:

Generally, a service should have a key of the strongest enctype that both it and the KDC support. If the KDC is running a release earlier than krb5-1.11, it is also useful to generate an additional key for each enctype that the service can support. The KDC will only use the first key in the list of long-term keys for encrypting the service ticket, but the additional long-term keys indicate the other enctypes that the service supports.

As noted above, starting with release krb5-1.11, there are additional configuration settings that control session key enctype selection independently of the set of long-term keys that the KDC has stored for a service principal.

Configuration variables:

The following [libdefaults] settings in krb5.conf will affect how enctypes are chosen.

allow_weak_crypto
defaults to false starting with krb5-1.8. When false, removes weak enctypes from permitted_enctypes, default_tkt_enctypes, and default_tgs_enctypes. Do not set this to true unless the use of weak enctypes is an acceptable risk for your environment and the weak enctypes are required for backward compatibility.

permitted_enctypes
controls the set of enctypes that a service will permit for session keys and for ticket and authenticator encryption. The KDC and other programs that access the Kerberos database will ignore keys of non-permitted enctypes. Starting in release 1.18, this setting also acts as the default for default_tkt_enctypes and defaut_tgs_enctypes.

default_tkt_enctypes
controls the default set of enctypes that the Kerberos client library requests when making an AS-REQ. Do not set this unless required for specific backward compatibility purposes; stale values of this setting can prevent clients from taking advantage of new stronger enctypes when the libraries are upgraded.

default_tgs_enctypes
controls the default set of enctypes that the Kerberos client library requests when making a TGS-REQ. Do not set this unless required for specific backward compatibility purposes; stale values of this setting can prevent clients from taking advantage of new stronger enctypes when the libraries are upgraded.

The following per-realm setting in kdc.conf affects the generation of long-term keys.

supported_enctypes
controls the default set of enctype-salttype pairs that kadmind will use for generating long-term keys, either randomly or from passwords

6.5 ELECTRONIC MAIL SECURITY

Email security is a term for describing different procedures and techniques for protecting email accounts, content, and communication against unauthorized access, loss or compromise. Email is often used to spread malware, spam and phishing attacks.

6.6 PRETTY GOOD PRIVACY

PGP is a remarkable phenomenon. Largely the effort of a single person, Phil Zimmermann, PGP provides a confidentiality and authentication service that can be used for electronic mail and file storage applications. In essence, Zimmermann has done the following:

- Selected the best available cryptographic algorithms as building blocks
- Integrated 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
- Made the package and its documentation, including the source code, freely available via the Internet, bulletin boards, and commercial networks such as AOL (America On Line)
- Entered into an agreement with a company (Viacrypt, now Network Associates) 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 platforms, including Windows,UNIX, Macintosh, and many more. In addition, the commercial version satisfies users who want a product that comes with vendor support.
- It is based on algorithms that have survived extensive public review and are considered extremely secure. Specifically, the package includes RSA, DSS, and Diffie-Hellman for public-key encryption; CAST-128, IDEA, and 3DES for symmetric encryption; and SHA-1 for hash coding.
- It has a wide range of applicability, from corporations that wish to select and enforce a standardized scheme for encrypting files and messages to individuals who wish to communicate securely with others worldwide over the Internet and other networks.

- It was not developed by, nor is it controlled by, any governmental or standards organization. For those with an instinctive distrust of "the establishment," this makes PGP attractive.
- PGP is now on an Internet standards track (RFC 3156). Nevertheless, PGP still has an aura of an antiestablishment endeavor.

We begin with an overall look at the operation of PGP. Next, we examine how cryptographic keys arecreated and stored. Then, we address the vital issue of public key management.

6.7 S/MIME

S/MIME (Secure/Multipurpose Internet Mail Extension) is a security enhancement to the MIME Internet email format standard, based on technology from RSA Data Security. Although both PGP and S/MIME are on an IETF standards track, it appears likely that S/MIME will emerge as the industry standard for commercial and organizational use, while PGP will remain the choice for personal e-mail security for many users. S/MIME is defined in a number of documents, most importantly RFCs 3369, 3370, 3850 and 3851.

To understand S/MIME, we need first to have a general understanding of the underlying e-mail format that it uses, namely MIME. But to understand the significance of MIME, we need to go back to the traditional e-mail format standard, RFC 822, which is still in common use. Accordingly, this section first provides an introduction to these two earlier standards and then moves on to a discussion of S/MIME.

RFC 822:

RFC 822 defines a format for text messages that are sent using electronic mail. It has been the standard for Internet-based text mail message and remains in common use. In the RFC 822 context, messages are viewed as having an envelope and contents. The envelope contains whatever information is needed to accomplish transmission and delivery. The contents compose the object to be delivered to the recipient. The RFC 822 standard applies only to the contents. However, the content standard includes a set of header fields that may be used by the mail system to create the envelope, and the standard is intended to facilitate the acquisition of such information by programs.

The overall structure of a message that conforms to RFC 822 is very simple. A message consists of some number of header lines (*the header*) followed by unrestricted text (*the body*). The header is separated from the body by a blank line. Put differently, a message is ASCII text, and all lines up to the first blank line are assumed to be header lines used by the user agent part of the mail system.

A header line usually consists of a keyword, followed by a colon, followed by the keyword's arguments; the format allows a long line to be broken up into several lines. The most frequently used keywords are *From*, *To*, *Subject*, and *Date*. Here is an example message:

Date: Tue, 16 Jan 1998 10:37:17 (EST)
From: "William Stallings" <ws@shore.net>
Subject: The Syntax in RFC 822
To: Smith@Other-host.com
Cc: Jones@Yet-Another-Host.com
Hello. This section begins the actual
message body, which is delimited from the
message heading by a blank line.

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. [RODR02] lists the following limitations of the SMTP/822 scheme:

- SMTP cannot transmit executable files or other binary objects. A number of schemes are in use for converting binary files into a text form that can be used by SMTP mail systems, including the popular UNIX UUencode/UUdecode scheme. However, none of these is a standard or even a de facto standard.
- 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.
- SMTP servers may reject mail message over a certain size.
- SMTP gateways that translate between ASCII and the character code EBCDIC do not use a consistent set of mappings, resulting in translation problems.
- SMTP gateways to X.400 electronic mail networks cannot handle nontextual data included inX.400 messages

6.8 DATA COMPRESSION USING ZIP

PGP makes use of a compression package called ZIP, written by Jean-lup Gailly, Mark Adler, and Richard Wales. ZIP is a freeware package written in C that runs as a utility on UNIX and some other systems.

ZIP is functionally equivalent to PKZIP, a widely available shareware package for Windows systemsdeveloped by PKWARE, Inc.

The zip algorithm is perhaps the most commonly used cross-platformcompression technique; freeware and shareware versions are available for Macintosh and other systemsas well as Windows and UNIX systems.

Zip and similar algorithms stem from research by Jacob Ziv and Abraham Lempel. In 1977, theydescribed a technique based on a sliding window buffer that holds the most recently processed text[ZIV77]. This algorithm is generally referred to as LZ77. A version of this algorithm is used in the zipcompression scheme (PKZIP, gzip, zipit, etc.).

LZ77 and its variants exploit the fact that words and phrases within a text stream (image patterns in thecase of GIF) are likely to be repeated. When a repetition occurs, the repeated sequence can be replacedby a short code. The compression program scans for such repetitions and develops codes on the fly toreplace the repeated sequence. Over time, codes are reused to capture new sequences. The algorithmmust be defined in such a way that the decompression program is able to deduce the current mappingbetween codes and sequences of source data.

6.9 RADIX-64 CONVERSION

Both PGP and S/MIME make use of an encoding technique referred to as radix-64 conversion. This technique maps arbitrary binary input into printable character output. The form of encoding has the following relevant characteristics:

The range of the function is a character set that is universally representable at all sites, not a specific binary encoding of that character set. Thus, the characters themselves can be encoded into whatever form is needed by a specific system. For example, the character "E" is represented in an ASCII-based system as hexadecimal 45 and in an EBCDIC-based system as hexadecimal C5.

The character set consists of 65 printable characters, one of which is used for padding. With 26 = 64 available characters, each character can be used to represent 6 bits of input.

No control characters are included in the set. Thus, a message encoded in radix 64 can traverse mail-handling systems that scan the data stream for control characters.

The hyphen character ("-")is not used. This character has significance in the RFC 822 format and should therefore be avoided.

5	F
6	G
7	H
8	I
9	J
10	K
11	L
12	M
13	N
14	O
15	P
16	Q
17	R
18	S
19	T
20	U
21	V
22	W
23	X
24	Y
25	Z
26	a
27	b
28	c
29	d
30	e
31	f
32	g
33	h
34	i
35	j
36	k
37	l
38	m

6-Bit	Character Encoding
0	A
1	B
2	C
3	D
4	E

39	n
40	o
41	p
42	q
43	r
44	s
45	t
46	u
47	v
48	w
49	x
50	y
51	z
52	0
53	1
54	2
55	3
56	4
57	5
58	6
59	7
60	8
61	9
62	+
63	/
(pad)	=

6.10 PGP RANDOM NUMBER GENERATION: -

PGP uses a complex and powerful scheme for generating random numbers and pseudorandom numbers,

for a variety of purposes. PGP generates random numbers from the content and timing of user

keystrokes, and pseudorandom numbers using an algorithm based on the one in ANSI X9.17. PGP uses

these numbers for the following purposes:
[Page 480]

● True random numbers:
used to generate RSA key pairs
provide the initial seed for the pseudorandom number generator
provide additional input during pseudorandom number generation

● Pseudorandom numbers:
used to generate session keys
used to generate initialization vectors (IVs) for use with the session key in CFB
mode encryption

True Random Numbers

PGP maintains a 256-byte buffer of random bits. Each time PGP expects a keystroke, it records the time,
in 32-bit format, at which it starts waiting. When it receives the keystroke, it records the time the key
was pressed and the 8-bit value of the keystroke. The time and keystroke information are used to
generate a key, which is, in turn, used to encrypt the current value of the random-bit buffer.

Pseudorandom Numbers

Pseudorandom number generation makes use of a 24-octet seed and produces a 16-octet session key,
an 8-octet initialization vector, and a new seed to be used for the next pseudorandom number
generation. The algorithm is based on the X9.17 algorithm described in Chapter 7 (see Figure 7.14) but
uses CAST-128 instead of triple DES for encryption. The algorithm uses the following data structures:

1.
Input
randseed.bin (24 octets): If this file is empty, it is filled with 24 true random
octets.
message: The session key and IV that will be used to encrypt a message are

themselves a function of that message. This further contributes to the randomness
of the key and IV, and if an opponent already knows the plaintext content of the
message, there is no apparent need for capturing the one-time session key.

2.

Output

K (24 octets): The first 16 octets, K[0..15], contain a session key, and the last
eight octets, K[16..23], contain an IV.

randseed.bin (24 octets): A new seed value is placed in this file.

3.

Internal data structures

dtbuf (8 octets): The first 4 octets, dtbuf[0..3], are initialized with the current
date/time value. This buffer is equivalent to the DT variable in the X12.17
algorithm.

rkey (16 octets): CAST-128 encryption key used at all stages of the algorithm.

rseed (8 octets): Equivalent to the X12.17 V*i* variable.

rbuf (8 octets): A pseudorandom number generated by the algorithm. This buffer
is equivalent to the X12.17 R*i* variable.

K' (24 octets): Temporary buffer for the new value of randseed.bin.

The algorithm consists of nine steps, G1 through G9. The first and last steps are obfuscation steps,
intended to reduce the value of a captured randseed.bin file to an opponent.

To summarize,

G1. [Prewash previous seed]

a. Copy randseed.bin to K[0..23].

b. Take the hash of the message (this has already been
generated if the message is being signed; otherwise the
first 4K octets of the message are used). Use the
result as a key, use a null IV, and encrypt K in CFB
mode; store result back in K.

G2. [Set initial seed]

a. Set dtbuf[0..3] to the 32-bit local time. Set
dtbuf[4..7] to all zeros. Copy rkey K[0..15]. Copy

rseed K[16..23].

b. Encrypt the 64-bit dtbuf using the 128-bit rkey in ECB mode; store the result back in dtbuf.

G3. [Prepare to generate random octets] Set rcount 0 and k 23. The loop of steps G4-G7 will be executed 24 times (k = 23...0), once for each random octet produced and placed in K. The variable rcount is the number of unused random octets in rbuf. It will count down from 8 to 0 three times to generate the 24 octets.

G4. [Bytes available?] If rcount = 0 goto G5 else goto G7. Steps G5 and G6 perform one instance of the X12.17 algorithm to generate a new batch of eight random octets.

[Page 482]

G5. [Generate new random octets]

a. rseed rseed dtbuf

b. rbuf Erkey[rseed] in ECB mode

G6. [Generate next seed]

a.rseed rbuf dtbuf

b. rseed Erkey[rseed] in ECB mode

c. Set rcount 8

G7. [Transfer one byte at a time from rbuf to K]

a. Set rcount rcount 1

b. Generate a true random byte b, and set K[k] rbuf[rcount] b

G8. [Done?] If k = 0 goto G9 else set k k 1 and goto G4

G9. [Postwash seed and return result]

a. Generate 24 more bytes by the method of steps G4-G7, except do not XOR in a random byte in G7. Place the result in buffer K'

b. Encrypt K' with key K[0..15] and IV K[16..23] in CFB mode; store result in randseed.bin

c. Return

6.11SUMMARY

This chapter is guiding about security and encryption techniques and covered almost all topics.

6.12REFERENCES AND BIBLIOGRAPHY

- Cryptography: Theory and Practice, Douglas Stinson, CRC Press, CRC Press LLC
- Cryptography and Network Security Principles and Practices, Fourth Edition, WilliamStallings, PHI(Pearson),

6.13 EXERCISE

1. What are the five principal services provided by PGP?
2. What is the utility of a detached signature?
3. Why does PGP generate a signature before applying compression?
4. What is R64 conversion?
5. Why is R64 conversion useful for an e-mail application?
6. Why is the segmentation and reassembly function in PGP needed?
7. How does PGP use the concept of trust?
8. What is RFC 822?
9. What is MIME?
10. What is S/MIME?

*****

UNIT V

7

IP SECURITY

Unit Structure

7.1 OBJECTIVE

The main objective is to understand various security protocols that are used in real life transactions. The basic knowledge of the protocols such as- TCP/IP etc and some transactions like-banking and online transactions which are carried out in real life.

7.2 INTRODUCTION

The applications of IPSec :

(i) Secure Remode Internet Access : Using IPSec, make a local call to our Internet Service Provider (ISP) so as to connect to our organization's network in a secure fashion from our home or hotel. From there, access the corporate network facilities or access remote desktops/servers.

(ii) Secure Branch Office Connectivity : Rather than subscribing to an expensive leased line for connecting its branches across cities/countries, an organization can set up an IPSec-enabled network to securely connect all its branches over the Internet.

(iii) Set Up Communication with Other Organizations: Just as IPSec allows connectivity between various branches of an organization; it can also be used to connect the networks of different organizations together in a secure and inexpensive fashion.

Following are the main advantages of IPSec.

- IPSec is transparent to the end users. There is no need for a user training, key issuance, or revocation.
- When IPSec is configured to work with a firewall, it becomes the only entry-exit point for all traffic, making it extra secure.
- IPSec works at the network layer. Hence, no changes are needed to the upper layers (application and transport).
- When IPSec is implemented in a firewall or a router, all the outgoing and incoming traffic gets protected.
- IPSec can allow traveling staff to have secure access to the corporate network.
- IPSec allows interconnectivity between branches/offices in a very inexpensive manner.

7.3 IPSEC PROTOCOLS

IPSec consists of two main protocols:

These two protocols are required for the following purposes.

- The Authentication Header (AH) protocol provides authentication, integrity, and an optional anti-replay service. The IPSec AH is a header in an IP packet, which contains a cryptographic checksum (like a message digest or hash) for the contents of the packet. The AH is simply inserted between the IP header and any subsequent packet contents. No changes are required to the data contents of the packet. Thus, security resides completely in the contents of the AH.

- **The Encapsulating Security Payload (ESP):** protocol provides data confidentiality. The ESP protocol also defines a new header to be inserted into the IP packet. ESP processing also includes the transformation of the protected data into an unreadable, encrypted format. Under normal circumstances, the ESP will be inside the AH. That is, encryption happens first and then authentication.

Authentication Header (AH) of IPSec.

AH Format

The Authentication Header (AH) provides support for data integrity and authentication of IP packets. The data integrity service ensures that data inside IP packets is not altered during the transit. The authentication service enables an end user or a computer system to authenticate the user or the application at the other end and decide to accept or reject packets, accordingly. This also prevents the IP spoofing attacks. AH is based on the MAC protocol, which means that the two communicating parties must share a secret key to use AH.

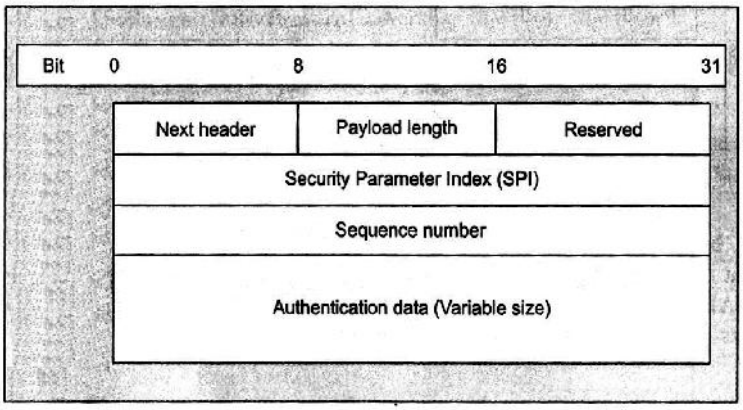


Fig.: Authentication Header (AH) format

Table 1 : Authentication Header Field Descriptions

Field	Description
Next header	This 8-bit field identifies the type of header that immediately follows the AH.
Payload length	This 8-bit field contains the length of the AH in 32-bit words minus 2.
Reserved	This 16-bit field is reserved for future use.
Security Parameter Index (SPI)	This 32-bit field is used in combination with the source and destination addresses as well as the IPSec protocol to uniquely identify the Security Association (SA) for the traffic to which a datagram belongs.
Sequence number	This 32-bit field is used to prevent replay attacks.
Authentication data	This variable-length field contains the authentication data, called as the Integrity Check Value (ICV), for the datagram. This value is the MAC, used for authentication and integrity purposes.

7.4 MODES OF OPERATION OF AH OF IPSEC

- **AH transport mode:** In the transport mode, the position of the Authentication Header (AH) is between the original IP header and the original TCP header of the IP packet.
- **AH tunnel mode:** In the tunnel mode, the entire original IP packet is authenticated, and the AH is inserted between the original IP header and a new outer IP header. The inner IP header contains the ultimate source and destination IP addresses, whereas the outer IP header possibly contains different IP addresses (e.g., IP addresses of the firewalls or other security gateways).

Encapsulating Security Payload (ESP) of IPSec.

ESP Format:

The Encapsulating Security Payload (ESP) protocol provides confidentiality and integrity of messages. ESP is based on symmetric key cryptography techniques. ESP can be used in isolation or it can be combined with AH. The ESP packet contains four fixed-length fields and three variable-length fields. Figure shows the ESP format.

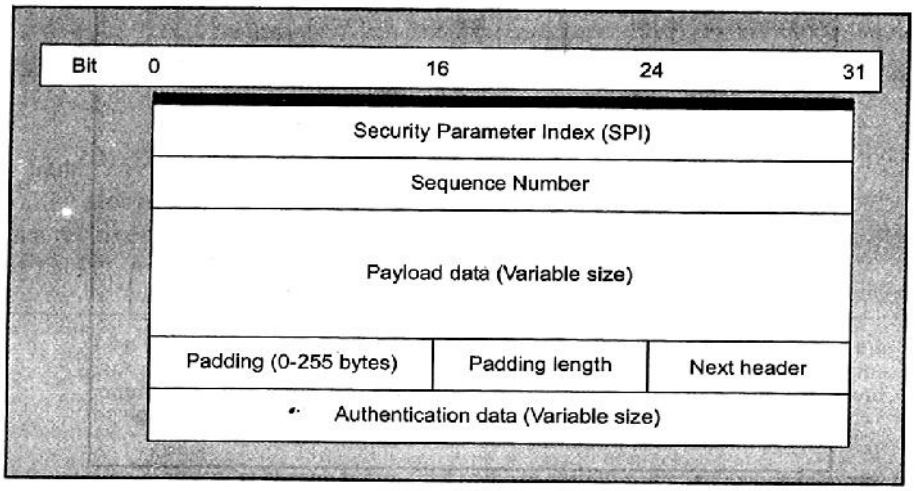


Fig: Encapsulating Security Payload (ESP) format

Table 1: ESP field descriptions

Field	Description
Security Parameter Index (SPI)	This 32-bit field is used in combination with the source and destination addresses as well as the IPSec protocol to uniquely identify the Security Association (SA) for the traffic to which a datagram belongs.
Sequence number	This 32-bit field is used to prevent replay attacks.
Payload data	This variable-length field contains the transport layer segment (transport mode) or IP packet

	(tunnel mode), which is protected by encryption.
Padding	This field contains the padding bits, if any. These are used by the encryption algorithm or for aligning the padding length field.
Padding length	This 8-bit field specifies the number of padding bytes in the immediately preceding field.
Next header	This 8-bit field identifies the type of encapsulated data in the payload.
Authentication data	This variable length field contains the authentication data, called as the Integrity Check Value (ICV), for the datagram.

7.5 MODES OF OPERATION OF ESP

(i) ESP Transport Mode : Transport mode ESP is used to encrypt and optionally authenticate the data carried by IP (for example, a TCP segment). Here, the ESP is inserted into the IP packet immediately before the transport layer header (i.e. TCP or UDP) and an ESP trailer (containing the fields Padding, Padding length and Next header) is added after the IP packet.

The operation of the ESP transport mode as follows :

- At the sender's end, the block of data containing the ESP trailer and the entire transport layer segment is encrypted and the plain text of this block is replaced with its corresponding cipher text to form the IP packet. Authentication is appended, if selected. This packet is now ready for transmission.
- The packet is routed to the destination. The intermediate routers need to take a look at the IP header as well as any IP extension headers, but not at the cipher text.
- At the receiver's end, the IP header plus any plain text IP extension headers are examined. The remaining portion of the packet is then decrypted to retrieve the original plain text transport layer segment.

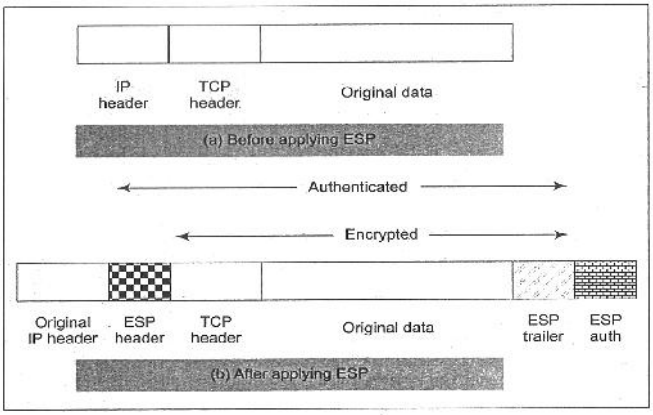


Fig. : ESP transport mode

(ii) ESP Tunnel Mode: The tunnel mode ESP encrypts an entire IP packet. Here, the ESP header is pre-fixed to the packet and then the packet along with the ESP trailer is encrypted. The IP header contains the destination address as well as intermediate routing information. Therefore, this packet cannot be transmitted as it is. Otherwise, the delivery of the packet would be impossible. Therefore, a new IP header is added, which contains sufficient information for routing.

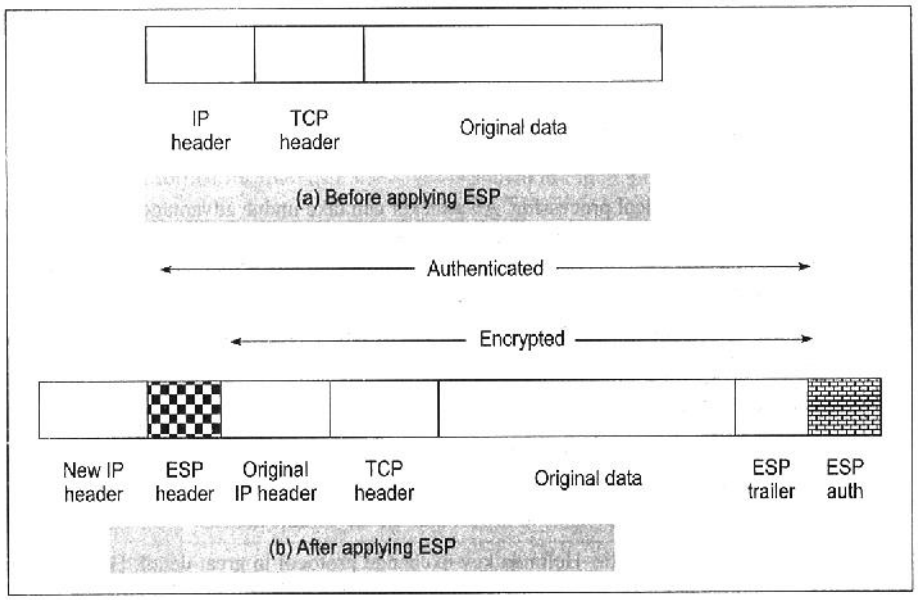


Fig. 2 : ESP tunnel mode

7. 6 INTERNET SECURITY PROTOCOLS:

They are SSL and SET.

7.7 SECURE SOCKET LAYER (SSL)

Handshake Protocol of SSL.

The handshake protocol is made up of four phases. These phases are:

- Establish security capabilities.
- Server authentication and key exchange
- Client authentication and key exchange
- Finish

Phase 1: Establish Security Capabilities:

This first phase of the SSL handshake is used to initiate a logical connection and establish the security capabilities associated with that connection. This consists of two message, the client hello and the server hello, as shown in figure.

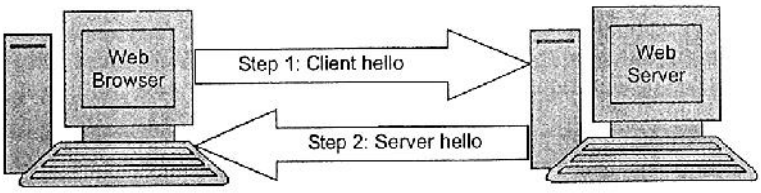


Fig. : SSL Handshake protocol Phase 1: Establish security capabilities

As shown in the figure, the process starts with a client hello message from the client to the server. It consists of the following parameters :

- **Version**: This field identifies the highest version of SSL that the client can support. At the time of this writing, this can be 2, 3, 1.
- **Random**: This field is useful for the later, actual communication between the client and the server. It contains two sub-fields :
- **Session id**: This is a variable-length session identifier. If this field contains a non-zero value, it means that there is already a connection between the client and the server.
- **Cipher suite**: This list contains a list of the cryptographic algorithms supported by the client (e.g. RSA, Diffie-Hellman, etc.
- **Compression method**: This field contains a list of the compression algorithms supported by the client.

The client sends the client hello message to the server and waits for the server's response. Accordingly, the server sends back a server hello message to the client. The server hello message consists of the following fields:

- **Version**: This field identifies the lower of the versions suggested by the client and the highest supported by the server. For instance, if the client had suggested version 3, but the server also supports version 3.1, the server will select 3.
- **Random**: This field has the same structure as the Random field of the client. However, the Random value generated by the server is completely independent of the client's Random value.
- **Session id**: If the session id value sent by the client was non-zero, the server uses the same value. Otherwise, the server creates a new session id and puts it in this field.
- **Cipher suite**: Contains a single cipher suite, which the server selects from the list sent earlier by the client.
- **Compression method**: Contains a compression algorithm, which the server selects from the list sent earlier by the client.

Phase 2: Server Authentication and Key Exchange:

The server initiates this second phase of SSL handshake and is the sole sender of all the messages in this phase. The client is the sole recipient of

all these messages. This phase contains four steps, as shown in figure. These steps are: Certificate, Server key exchange, Certificate request and Server hello done.

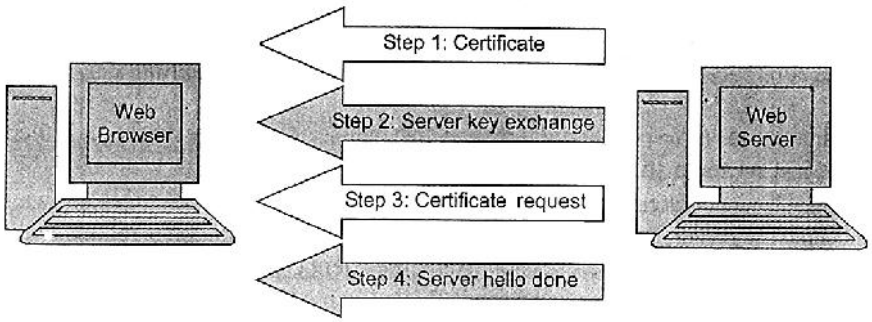


Fig 3. : SSL Handshake protocol Phase 2: Server authentication and key exchange

The four steps of this phase are as follows :

In the first step (certificate) : The server sends its digital certificate and the entire chain leading upto root CA to the client. This will help the client to authenticate the server using the server's public key from the server's certificate.

The second step (Server key exchange) : Is optional. It is used only if the server does not send its digital certificate to the client in step 1 above.

The third step (certificate request) : The server can request for the client's digital certificate. The client authentication in SSL is optional and the server may not always expect the client to be authenticated.

The last step (server hello done) : Message indicates to the client that its portion of the hello message (i.e. the server hello message) is complete.

Phase 3 : Client Authentication and Key Exchange:

The client initiates this third phase of the SSL handshake. The server is the sole recipient of all these messages. This phase contains three steps, as shown in figure. These steps are: Certificate, Client key exchange and Certificate verify.

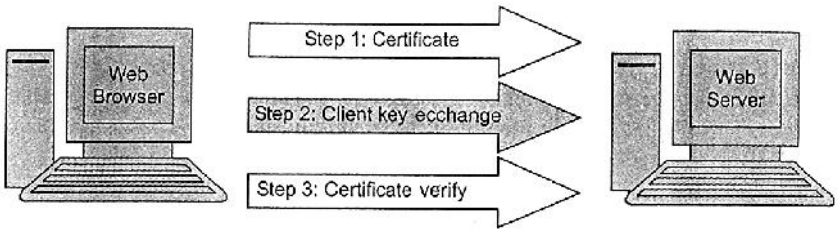


Fig : SSL Handshake protocol Phase 3: Client authentication and key exchange

- **The first step (certificate):** is optional. This step is performed only if the server had requested for the client's digital certificate. If the server has requested for the client's certificate and if the client does not have one, the client sends a No certificate message, instead of a Certificate message. It then is up to the server to decide if it wants to still continue or not.

- **The second step (client key exchange):** allows the client to send information to the server, but in the opposite direction. This information is related to the symmetric key. Here, the client creates a 48-bytes pre-master secret, and encrypts it with the server's public key and sends this encrypted pre-master secret to the server.

- **The third step (Certificate verify):** is necessary only if the server had demanded client authentication. If this is the case, the client has already sent its certificate to the server. However, additionally, the client also needs to prove to the server that it is the correct and authorized holder of the private key corresponding to the certificate.

Phase 4: Finish:

The client initiates this fourth phase of the SSL handshake, which the server ends. This phase contains four steps, as shown in figure. The first two messages are from the client : Change cipher specs Finished. The server responds back with two identical messages : Change cipher specs, Finished.

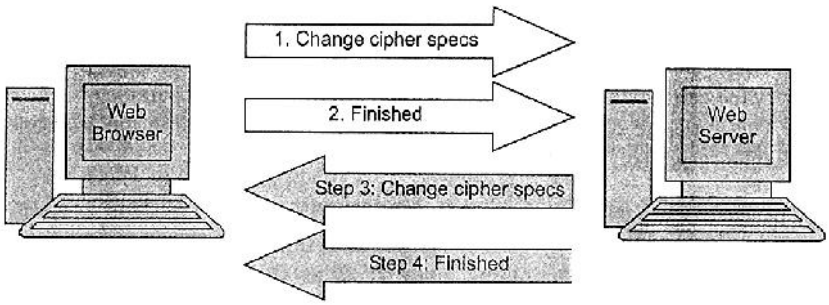


Fig. : SSL Handshake protocol Phase4: Finished

Record Protocol of SSL:

The Record Protocol:

The Record Protocol in SSL comes into picture after a successful handshake is completed between the client and the server. That is, after the client and the server have optionally authenticated each other and have decided what algorithms to use for secure information exchange. This protocol provides two services to an SSL connection, as follows:

- **Confidentiality:** This is achieved by using the secret key that is defined by the handshake protocol.

- **Integrity:** The handshake protocol also defines a shared secret key (MAC) that is used for assuring the message integrity.

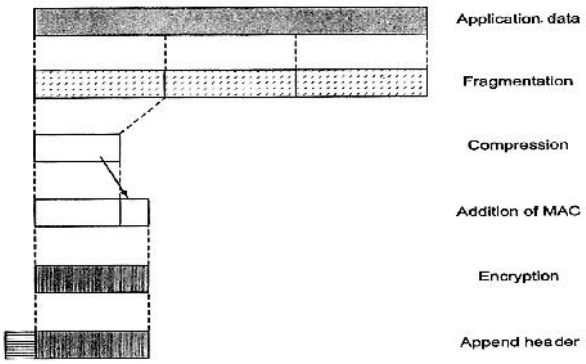


Fig. : SSL record protocol

As the figure shows, the SSL record protocol takes an application message as input. First, it fragments it into smaller blocks, optionally compresses each block, adds MAC, encrypts it, adds a header and gives it to the transport layer, where the TCP protocol processes it like any other TCP block. At the receiver's end, the header of each block is removed; the block is then decrypted, verified, decompressed, and reassembled into application messages. Steps in more detail as follows :

(c) Fragmentation : The original application message is broken into blocks, so that the size of each block is less than or equal to 2^{14} bytes (16,384 bytes).

(d) Compression : The fragmented blocks are optionally compressed. The compression process must not result into the loss of the original data, which means that this must be a lossless compression mechanism.

(e) Addition of MAC : Using the shared secret key established previously in the handshake protocol, the Message Authentication Code (MAC) for each block is calculated. This operation is similar to the HMAC algorithm.

(f) Encryption : Using the symmetric key established previously in the handshake protocol, the output of the previous step is now encrypted. This encryption may not increase the overall size of the block by more than 1024 bytes.

(g) Append Header : Finally, a header is added to the encrypted block.

Alert Protocol of SSL:

When either the client or the server detects an error, the detecting party sends an alert message to the other party. If the error is fatal, both the parties immediately close the SSL connection (which means that the transmission from, both the end is terminated immediately). Both the parties also destroy the session identifiers, secrets and keys associated with this connection before it is terminated.

Each alert message consists of two bytes. The first byte signifies the type of error. If it is a warning, this byte contains 1. If the error is fatal, this byte contains 2. The second byte specifies the actual error.

7.8 SECURE HYPER TEXT TRANSFER PROTOCOL (SHTTP)

The Secure Hyper Text Transfer Protocol (SHTTP) is a set of security mechanisms defined for protecting the Internet traffic. This includes the data entry forms and Internet- based transactions. SSL has become highly successfulZSHTTP has not. SHTTP works at the application layer, and is therefore, tightly coupled with HTTP.

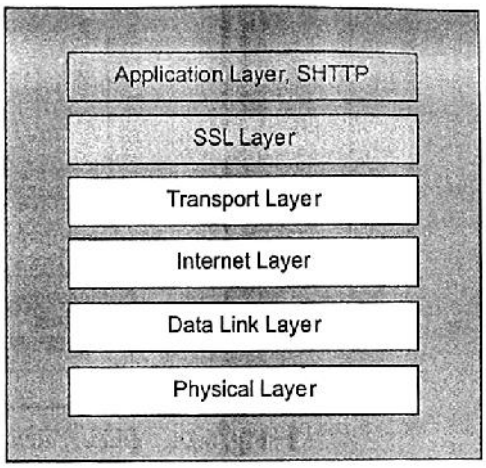


Fig: Positions of SHTTP and SSL in TCP/IP protocol suite

SHTTP supports both authentication and encryption of HTTP traffic between the client and the server. The encryption and digital signature formats used in SHTTP have origins in the PEM protocol.

The key difference between SSL and SHTTP is that SHTTP works at the level of individual messages. It can encrypt and sign individual messages. On the other hand, SSL does not differentiate between different messages.

7.9 SECURE ELECTRONIC TRANSACTION (SET):

Participants.

(a) Cardholder: A cardholder is an authorized holder of a payment card such as MasterCard or Visa that has been issued by an **issuer**.

(b) Merchant: A merchant is a person or an organization that wants to sell goods or services to cardholders.

(c) Issuer: The **issuer** is a financial institution (such as a bank) that provides a payment-card to a cardholder.

(d) Acquirer: This is a financial institution that has a relationship with merchants for processing payment-card authorizations and payments.

(e) Payment Gateway: The payment gateway processes the payment messages on behalf of the merchant. Specifically in SET, the payment gateway acts as an interface-between SET and the existing card-payment networks for payment authorizations.

(f) Certification Authority (CA): This is an authority that is trusted to provide public key certificates to cardholders, merchants and payment gateways. In fact, CAs are very crucial to the success of SET.

SET Processí:

(a) The Customer Opens an Account: The customer opens a credit-card account (such as MasterCard or Visa) with a bank (issuer) that supports electronic payment mechanisms and the SET protocol.

(b) The Customer Receives a Certificate: After the customer's identity is verified (with the help of details such as passport, business documents, etc.), the customer receives a digital certificate from a CA.

(c) The Merchant Receives a Certificate: A merchant that wants to accept a certain brand of credit cards must possess a digital certificate.

(d) The Customer Places an Order: This is a typical shopping-cart process wherein the customer browses the list of items available, searches for specific items, selects one or more of them, and places the order.

(e) The Merchant is Verified: The merchant also sends its digital certificate to the customer. This assures the customer that he/she is dealing with a valid merchant.

(f) The Order and Payment Details are sent: The customer sends both the order and payment details to the merchant along with the customer's digital certificate. The order confirms the purchase transaction with reference to the items mentioned in the order form.

(g) The Merchant Requests Payment Authorization: The merchant forwards the payment details sent by the customer to the payment gateway via the acquirer (or to the acquirer if the acquirer also acts as the payment gateway) and requests the payment gateway to authorize the payment (i.e. ensure that the credit card is valid and that the credit limits are not breached).

(h) The Payment Gateway Authorizes the Payment: Using the credit-card information received from the merchant, the payment gateway verifies the details of the customer's credit card with the help of the issuer, and either authorizes or rejects the payment.

(i) The Merchant Confirms the Order: Assuming that the payment gateway authorizes the payment, the merchant sends a confirmation of the order to the customer.

(j) The Merchant Provides Goods or Services: The merchant now ships the goods or provides the services as per the customer's order.

(k) The Merchant Requests Payment: The payment gateway receives a request from the merchant for making the payment. The payment gateway interacts with the various financial institutions such as the issuer, acquirer and the clearing house to effect the payment from the customer's account to the merchant's account.

7.10 SSL VERSUS SET

Comparison between SSL Versus SET:

Issue	SSL	SET
Main aim	Exchange of data in an encrypted form	E-commerce related payment mechanism
Certification	Two parties exchange certificates	All the involved parties must be certified by a trusted third party
Authentication	Mechanisms in place, but not very strong	Strong mechanisms for authenticating all the parties involved
Risk of merchant fraud	Possible, since customer gives financial data to merchant	Unlikely, since customer gives financial data to payment gateway
Risk of customer fraud	Possible, no mechanisms exist if a customer refuses to pay later	Customer has to digitally sign payment instructions
Action in case of customer fraud	Merchant is liable	Payment gateway is liable
Practical usage	High	Low at the moment, expected to grow

7.11 PRIVACY ENHANCED MAIL (PEM) PROTOCOL.

Introduction:

The Privacy Enhanced Mail (PEM) is an email security standard. PEM supports the three main cryptographic functions of encryption, non-repudiation, and message integrity.

How PEM Works?; PEM starts with a canonical conversion, which is followed by digital signature, then by encryption and finally, Base-64 encoding.

Step 1 (Canonical Conversion): There is a distinct possibility that the sender and the receiver of an email message use computers that have different architectures and operating systems. PEM transforms each email message into an abstract, canonical representation.

Step 2 (Digital Signature): This is a typical process of digital signature. It starts by creating a message digest of the email message using an algorithm such as MD2 or MD5, as shown in Fig.

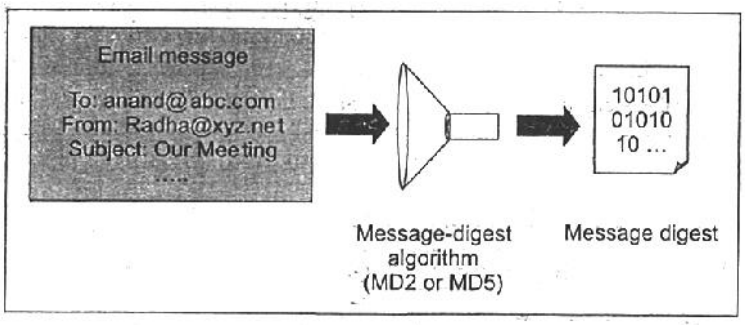


Fig: Message-digest creation of the original email message.

The message digest thus created is then encrypted with the sender's private key to form the sender's digital signature. This process is shown in Fig.

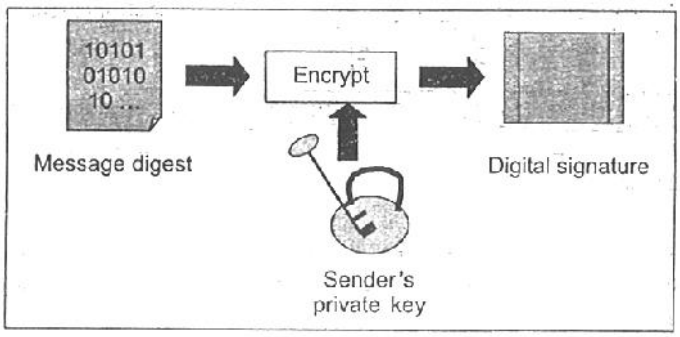


Fig: Creation of the sender's digital signature over the email message.

Step 3 (Encryption): In this step, the original email and the digital signature are encrypted together with a symmetric key. For this, the DES or DES-3 algorithm in CBC mode is used. This is shown in Fig.

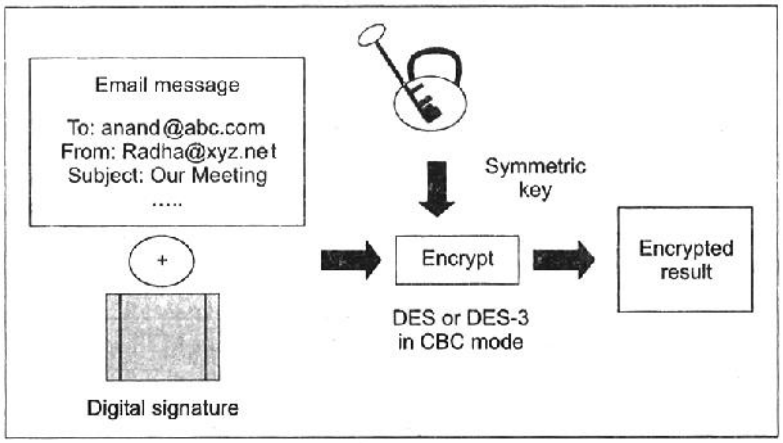


Fig: Encryption in PEM

Step 4 (Base-64 Encoding): This is the last step in PEM. The Base-64 encoding (also called Radix-64 encoding or ASCII armour) process transforms arbitrary binary input into printable character output. In this technique, the binary input is processed in blocks of 3 octets, or 24 bits. These 24 bits are 'considered to be made up of 4 sets, each of 6 bits. Each such set of 6 bits is mapped into an 8-bit output character in this process. This concept is shown in Fig.

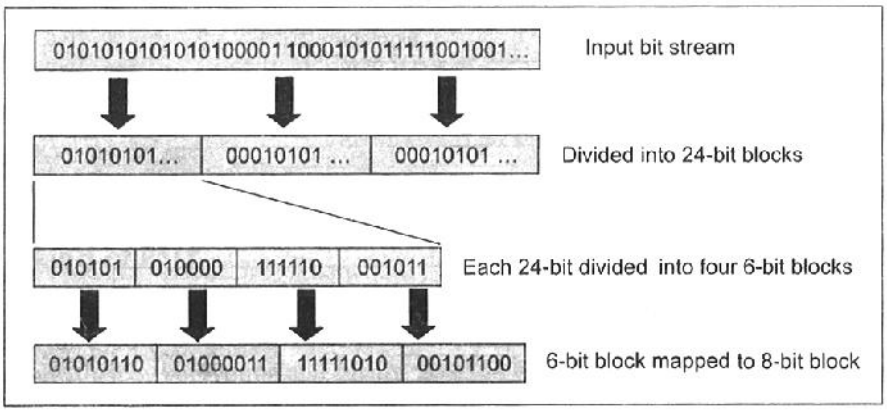


Fig: Base-64 encoding concept

7.12 PRETTY GOOD PRIVACY (PGP) PROTOCOL.

(A) The Working of PGP :

In PGP, the sender of the message needs to include the identifiers of the algorithm used in the message, along with the value of the keys. PGP starts with a digital signature, which is followed by compression, then by encryption, then by digital enveloping and finally, by Base-64 encoding.

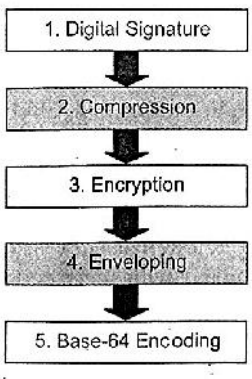


Fig: PGP operations

Note that the receiver has to perform these four steps in the reverse direction to retrieve the original plain text email message.

Step 1 (Digital Signature): This is a typical process of digital signature. In PGP, it consists of the creation of a message digest of the email message using the SHA**-1** algorithm. The resulting message digest is then encrypted with the sender's private key. The result is the sender's digital signature.

Step 2 (Compression): Here, the input message as well as the digital signature are compressed together to reduce the size of the final message that will be transmitted. For this, the famous ZIP program is used. ZIP is based on the **Lempel-Ziv algorithm.**

The Lempel-Ziv algorithm looks for repeated strings or words, and stores them in variables. It then replaces the actual occurrence of the repeated word or string with a pointer to the corresponding variable. For instance, consider the following string: What is your name? My name is Atul.

Using the Lempel-Ziv algorithm, two variables are used & also points A & B. This is shown in Fig.

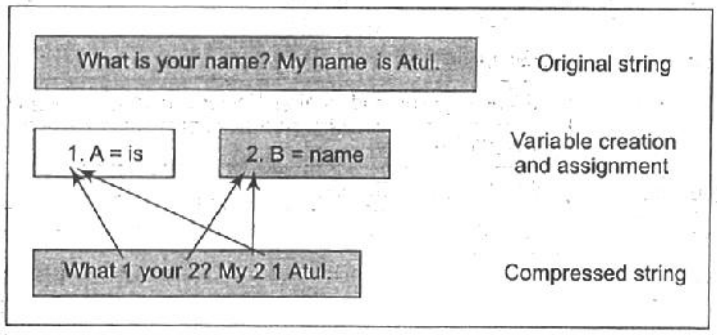


Fig: Lempel-Ziv algorithm, as used by the ZIP program

The resulting string What 1your 2? My 2 1 Atul. Is smaller compared to the original string What is your name? My name is Atul. Of course, the bigger the original string, the better the compression gets. The same process works for PGP.

Step 3 (Encryption): In this step, the compressed output of step 2 (i.e. the compressed form of the original email and the digital signature together) are encrypted with a symmetric key. For this, generally the IDEA algorithm in CFB mode is used.

Step 4 (Digital Enveloping): In this case, the symmetric key used for encryption in step 3 is now encrypted with the receiver's public key. The output of step 3 and step 4 together form a digital envelope.

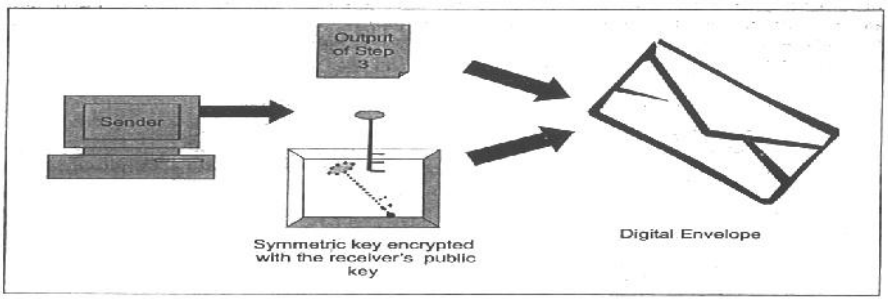


Fig: Formation of digital envelope

Step 5 (Base-64 encoding): The output of step 4 is Base-64 is encoded now.

7.13 SECURE MULTIPURPOSE INTERNET MAIL EXTENSIONS (S/MIME).

The Multipurpose Internet Mail Extensions (MIME) system extends the basic email system by permitting users to send binary files using the basic email system. The MIME specification adds five new headers to the email system, which describe information about the body of the message.

These headers are described as follows:

- **MIME-Version :** This contains the MIME version number. Currently, it has a value of 1.1. This field is reserved for the future use, when newer versions of MIME are expected to emerge. This field indicates that the message conforms to RFCs 2045 and 2046.
- **Content-Type:** Describes the data contained in the body of the message. The details provided are sufficient so that the receiver email system can deal with the received email message in an appropriate manner. The contents are specified as:

Type/Sub-type

MIME specifies 7 content types, and 15 content sub-types. Table 1 lists these types and sub-types.

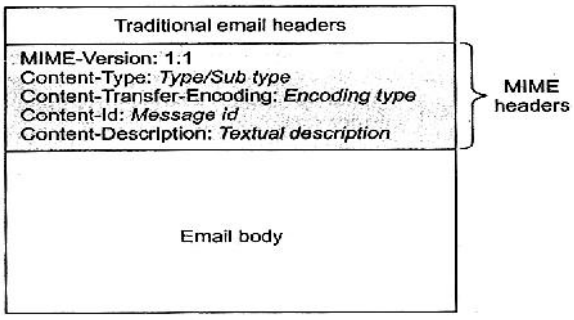


Fig: MIME headers in an email message

- **Content-Transfer-Encoding:** Specifies the type of transformation that has been used to represent the body of the message. In other words, the method used to encode the messages into zeroes and ones is defined here. There are five content encoding methods, as shown in Table 3.
- **Content-ID:** Identifies the MIME entities uniquely with reference to multiple contexts.
- **Content-Description:** Used when the body is not readable (e.g. video).

7.14 SUMMARY

Corporate networks can be attacked from outside or internal information can be leaked out. Encryption cannot prevent outside attackers from attacking a network. IPSec provides security between the transport and the Internet layers.

- IPSec provides authentication and confidentiality services.
- IPSec can be implemented in tunnel mode or transport mode.
- In the transport mode, the IP datagram except its header is encrypted by IPSec,

7.15 REFERENCE FOR FURTHER READING

Notes has been taken from : Introduction to Network Security by Atul Kahate.

7.16 UNIT END EXERCISES

1. List all Applications and Advantages of IPSec.
2. Describe IPSec Protocols.
3. Draw & Explain Authentication Header (AH) of IPSec.
4. Explain Modes of Operation of AH of IPSec.
5. Draw and explain Encapsulating Security Payload (ESP) of IPSec.
6. Describe Modes of Operation of ESP.
7. Write a short note on Virtual Private Networks (VPN).
8. Describe the Handshake Protocol of SSL.
9. Explain the Alert Protocol of SSL.
10. Describe Secure Hyper Text Transfer Protocol (SHTTP).
11. List all SET Participants.
12. Explain the SET Process.
13. Compare SSL Versus SET.
14. Write a short note on Privacy Enhanced Mail (PEM) Protocol.
15. Write a short note on Pretty Good Privacy (PGP) Protocol.
16. Write a short note on Secure Multipurpose Internet Mail Extensions (S/MIME).

UNIT VI

8

INTRUDERS, FIREWALL

Unit Structure

8.1 OBJECTIVE

In this chapter we will study about all the different types of virus, worms, intruders, firewall and learn about the configurations.

8.2 INTRODUCTION

In this chapter topics discussed are : virus, worms, trojans, Firewall and its type, DDOs attack.

8.3 INTRUDERS/INTRUSION

Intruders are said to be of three types:

- **Masquerader:** A user who does not have the authority to use a computer, but penetrates into a system to access a legitimate user's account is called as a masquerader. It is generally an external user.
- **Misfeasor:** There are two possible cases for an internal user to be called as a misfeasor.
  - A legitimate user, who does not have access to some applications, data or resources accesses them.

- Z A legitimate user, who has access to some applications, data or resources misuses these privileges.

-) **Clandestine user:** An internal or external user who tries to work using the privileges of a supervisor user to avoid auditing information being captured and recorded is called as to clandestine user.

Honeypots:

Modern intrusion detection systems make use of a novel idea, called as honeypots. A honeypot is a trap that attracts potential attackers. A honeypot is designed to do the following:

-) Divert the attention of a potential intruder from critical systems.
-) Collect information about the intruder's actions.
-) Provide encouragement to the intruder to stay on for some time, allowing the administrators to detect this and swiftly act on it.

Honeypots are designed with two important goals in mind:

(a) Make them look like real-life systems. Put as much of real-looking (but fabricated) information into them as possible.

(b) Do not allow legitimate users to know about or access them.

8.4 VIRUS

A virus is a piece of program code that attaches itself to legitimate program code, and runs when the legitimate program runs. It can then infect other programs in that computer, or programs that are in other computers but on the same network. After deleting all the files from the current user's computer, the virus self-propagates by sending its code to all users whose email addresses are stored in the current user's address book.

Viruses can also be triggered by specific events (e.g. a virus could automatically execute at 12 p.m. every day). Usually viruses cause damage to computer and network systems to the extent that they can be repaired, assuming that the organization deploys good backup and recovery procedures.

A virus is a computer program that attaches itself to another legitimate program and causes damage to the computer system or to the network.

A. Virus goes through four phases:

(a) Dormant phase: Here, the virus is idle. It gets activated based on certain action or event (e.g. the user typing a certain key or certain date or time is reached, etc.). This is an optional phase.

(b) Propagation phase: In this phase, a virus copies itself and each copy starts creating more copies of self, thus propagating the virus.

(c) Triggering phase: A dormant virus moves into this phase when the action/event for which it was waiting is initiated.

(d) Execution phase: This is the actual work of the virus, which could be harmless (display some message on the screen) or destructive (delete a file on the disk).

Viruses can be classified into the following categories:

(a) Parasitic Virus: This is the most common form of virus. Such a virus attaches itself to executable files and keeps replicating. Whenever the infected file is executed, the virus looks for other executable files to attach itself and spread.

(b) Memory-resident Virus: This type of virus first attaches itself to an area of the main memory and then infects every executable program that is executed.

(c) Boot sector Virus: This type of virus infects the master boot record of the disk and spreads on the disk when the operating system starts booting the computer.

(d) Stealth Virus: This virus has intelligence built in, which prevents anti-virus software programs from detecting it.

(e) Polymorphic Virus: A virus that keeps changing its signature (i.e. identity) on every execution, making it very difficult to detect.

(f) Metamorphic Virus: In addition to changing its signature like a polymorphic virus, this type of virus keeps rewriting itself every time, making its detection even harder.

8.5 OTHER RELATED THREATS -WORM

A worm, however, does not modify a program. Instead, it replicates itself again and again. The replication grows so much that ultimately the computer or the network on which the worm resides, becomes very slow, ultimately coming to a halt. Thus, the basic purpose of a worm attack is different from that of a virus. A worm attack attempts to make the computer or the network under attack unusable by eating all its resources.

A worm does not perform any destructive actions, and instead, only consumes system resources to bring it down.

Trojan horse:

A Trojan horse is a hidden piece of code, like a virus. However, the purpose of a Trojan horse is different. Whereas the main purpose of a virus is to make some sort of modifications to the target computer or

network, a Trojan horse attempts to reveal confidential information to an attacker.

A Trojan horse could silently sit in the code for a Login screen by attaching itself to it. When the user enters the user id and password, the Trojan horse could capture these details, and send this information to the attacker without the knowledge of the user who had entered the id and password. The attacker can then merrily misuse the user id and password to gain access to the system.

A Trojan horse allows an attacker to obtain some confidential information about a computer or a network.

8.6 VIRUS COUNTERMEASURES

Do not allow a virus to get into the system in the first place, or block the ability of a virus to modify any files containing executable code or macros. The count measures are:

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 Remove the vir virus from all infected systems so that the virus cannot spread further.

8.7 DISTRIBUTED DENIAL OF SERVICE ATTACKS

It is a cyber attack where in which the attacker seeks to make a system or network resource unavailable to its intended users by temporarily disrupting services of a host connected to the network.

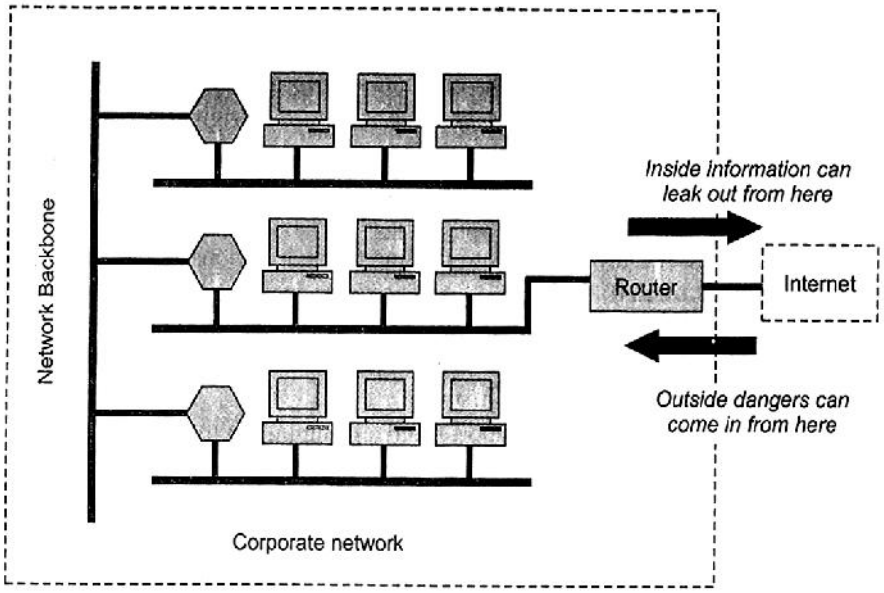
In a distributed denial-of-service attack (DDoS attack), the incoming traffic flooding the victim originates from many different sources. This effectively makes it impossible to stop the attack simply by blocking a single source.

DDOS Stands for Distribute n a distributed denial-of-service attack (DDoS attack), the incoming traffic flooding the victim originates from many different sources. This effectively makes it impossible to stop the attack simply by blocking a single source.d Denial of service attack where multiple system attacks the victims systems. Victim PC is loaded from the packet of data sent from Multiple location. Bots are used to attack at the same time. They are difficult to trace.

8.8 FIREWALLS

At a broad level, there are two kinds of attacks:

- Most corporations have large amounts of valuable and confidential data in their networks. Leaking of this critical information to competitors can be a great setback.
- Apart from the danger of the insider information leaking out, there is a great danger of the outside elements (such as viruses and worms) entering a corporate network.



Threats from inside and outside a corporate network

Encryption of information (if implemented properly) renders its transmission to the outside world redundant. That is, even if confidential information flows out of a corporate network, if it is in encrypted form, outsiders cannot make any sense of it. However, encryption does not work in the other direction. Outside attackers can still try to break inside a corporate network.

If implemented, it guards a corporate network by standing between the network and the outside world. All traffic between the network and the Internet in either direction must pass through the firewall. The firewall decides if the traffic can be allowed to flow or whether it must be stopped from proceeding further.

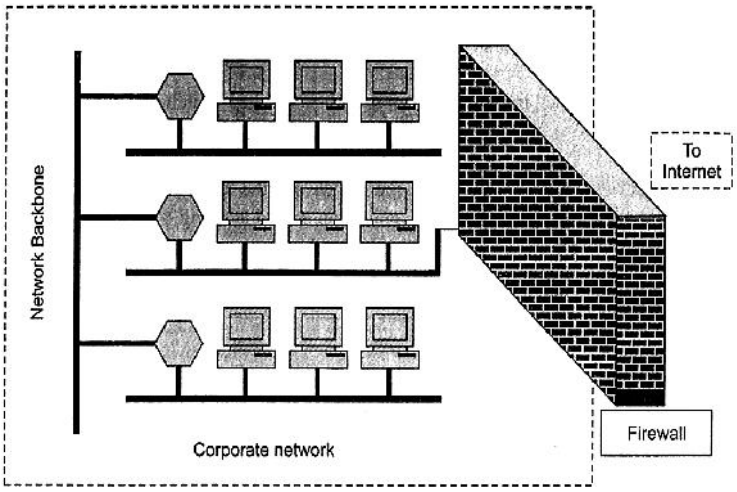


Fig.1:Firewall

Of course, technically, a firewall is a specialized version of a router. Apart from the basic routing functions and rules, a router can be configured to perform the firewall functionality, with the help of additional software resources.

The characteristics of a good firewall implementation can be described as follows.

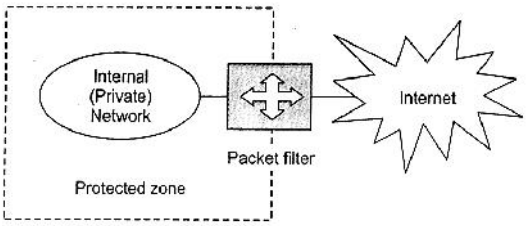
- All traffic from inside to outside and vice versa, must pass through the firewall. To achieve this, all the access to the local network must first be physically blocked and access only via the firewall should be permitted.
- Only the traffic authorized as per the local security policy should be allowed to pass through.
- The firewall itself must be strong enough, so as to render attacks on it useless.

8.9 FIREWALL DESIGN

Packet Filters:

A packet filter applies a set of rules to each packet and based on the outcome, decides to either forward or discard the packet. It is also called as screening router or screening filter.

Such a firewall implementation involves a router, which is configured to filter packets going in either direction (from the local network to the outside world and vice versa). The filtering rules are based on a number of fields in the IP and TCP/UDP headers, such as source and destination IP addresses, IP protocol field.



Packet filter

A packet filter performs the following functions:

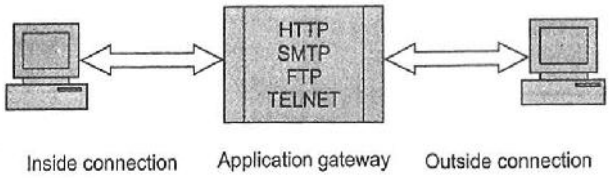
(a) Receive each packet as it arrives.

(b) Pass the packet through a set of rules, based on the contents of the IP and transport header fields of the packet. If there is a match with one of the set rules, decide whether to accept or discard the packet based on that rule.

(c) If there is no match with any rule, take the default action. The default can be discard all packets or accept all packets.

Application Gateways:

An application gateway is also called a proxy server. This is because it acts like a proxy and decides about the flow of application level traffic.



Application gateway

Application gateways typically work as follows.

(i) An internal user contacts the application gateway using a TCP/IP application, such as HTTP or TELNET.

(ii) The application gateway asks the user about the remote host with which the user wants to set up a connection for actual communication.

The application gateway also asks for the user id and the password required to access the services of the application gateway.

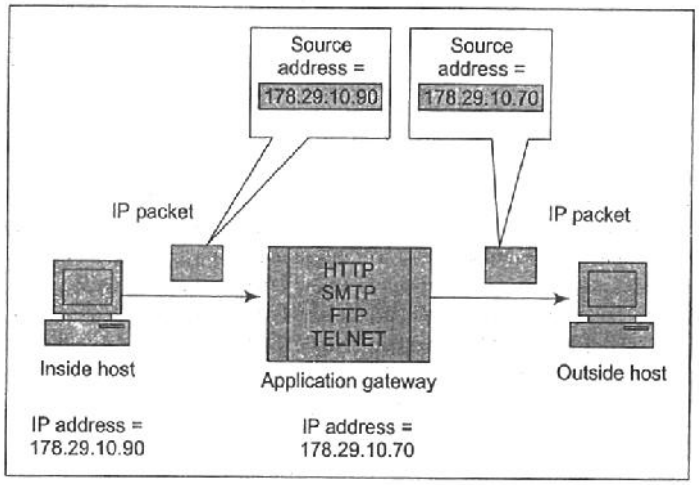
(iii) The user provides this information to the application gateway.

(iv) The application gateway now accesses the remote host on behalf of the user and passes the packets of the user to the remote host.

A circuit gateway, creates a new connection between itself and the remote host. The user is not aware of this and thinks that there is a direct connection between itself and the remote host. Also, the circuit gateway changes the source IP address in the packets from the end

user's IP address to its own. This way, the IP addresses of the computers of the internal users are hidden from the outside world.

The SOCKS server is an example of the real-life implementation of a circuit gateway.

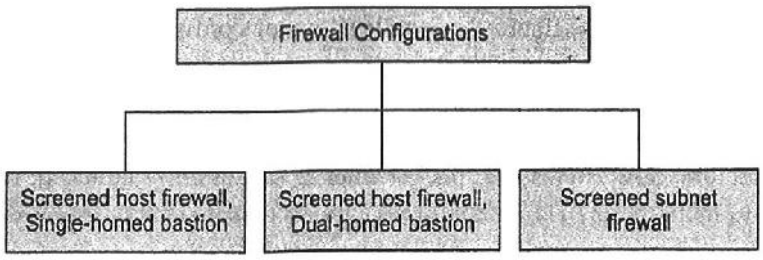


Circuit-gateway operation

(v) From here onwards, the application gateway acts like a proxy of the actual end user and delivers packets from the user to the remote host and vice versa.

Firewall Configuration:

There are three possible configurations of firewalls.

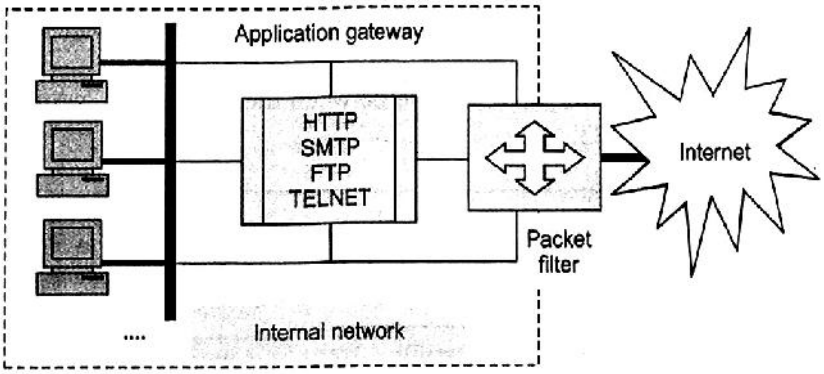


Firewall configurations

Screened Host Firewall, Single-Homed Bastion:

In the Screened host firewall, Single-homed bastion configuration, a firewall set up consists of two parts: a packet-filtering router and an application gateway. Their purposes are as follows.

- The packet filter ensures that the incoming traffic (i.e. from the Internet to the corporate network) is allowed only if it is destined for the application gateway, by examining the destination address field of every incoming IP packet. Similarly, it also ensures that the outgoing traffic (i.e. from the corporate network to the Internet) is allowed only if it is originating from the application gateway, by examining the source address field of every outgoing IP packet.



Screened host firewall, Single-homed bastion

Screened Host Firewall, Dual-Homed Bastion:

Here, direct connections between the internal hosts and the packet filter are avoided. Instead, the packet filter connects only to the application gateway, which, in turn, has a separate connection with the internal hosts. Therefore, now even if the packet filter is successfully attacked, only the application gateway is visible to the attacker. The internal hosts are protected.

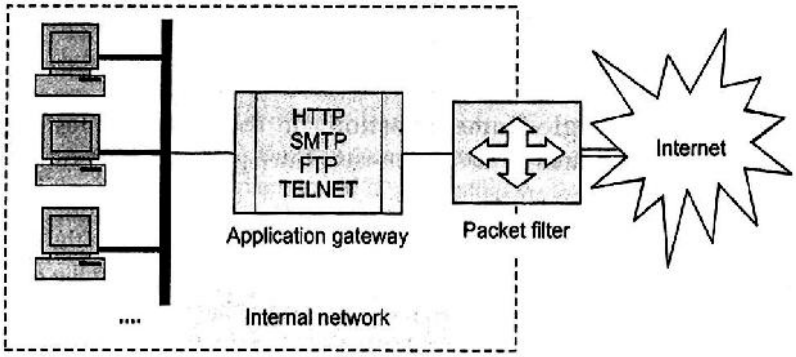


Fig. 3 :Screened host firewall, Dual-homed bastion

Screened Subnet Firewall:

The Screened subnet firewall offers the highest security among the possible firewall configurations. Here, two packet filters are used, one between the Internet and the application gateway, as previously and another one between the application gateway and the internal network.

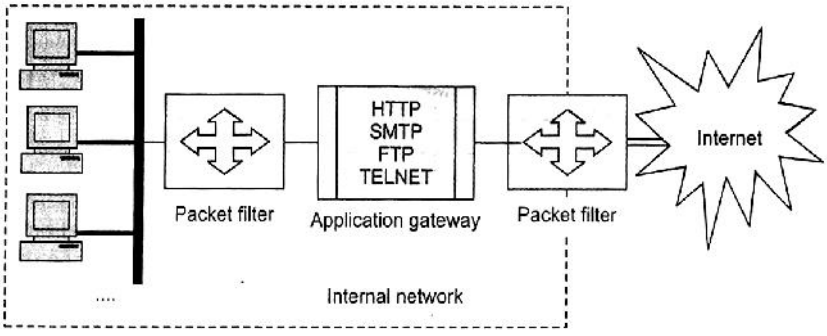


Fig. 4 :Screened subnet firewall

8.10 TRUSTED SYSTEMS SECURITY EVALUATION CRITERIA

The United States Department of Defense published a series of documents to classify the security of operating systems, known as the *Trusted Systems Security Evaluation Criteria* .

TCSEC was developed to meet three objectives:

- To give users a yardstick for assessing how much they can trust computer systems for the secure processing of classified or other sensitive information
- To guide manufacturers in what to build into their new, widely available commercial products to satisfy trust requirements for sensitive applications.
- To provide a basis for specifying security requirements for software and hardware
  Acquisitions Table provides a brief overview of the different classification levels.

8.11 PASSWORDS

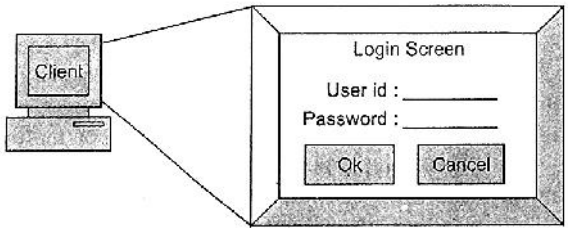
Passwords are the most common form of authentication. A password is a string of alphabets, numbers, and special characters, which is supposed to be known only to the entity (usually a person) that is being authenticated.

Clear Text-Password Mechanism:
Its Working:

This is the simplest password-based authentication mechanism. Every user in the system is assigned a user id and an initial password. The password is stored in clear text in the user database against the user id on the server. The authentication mechanism works as follows:

Step 1: Prompt for user id and password:

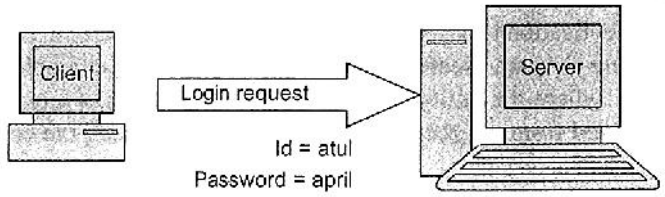
During authentication, the application sends a screen to the user, prompting for the user id and password.



Prompt for user id and password.

Step 2:User enters user id and password:

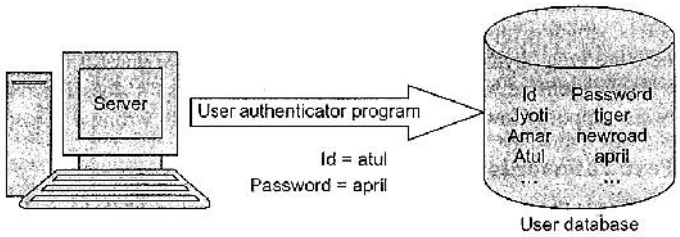
The user enters, id and password and click onOK button. This causes the user id and password to travel in clear text to the server.



User id and password travel in clear text to the server.

Step 3:User id and password validation:

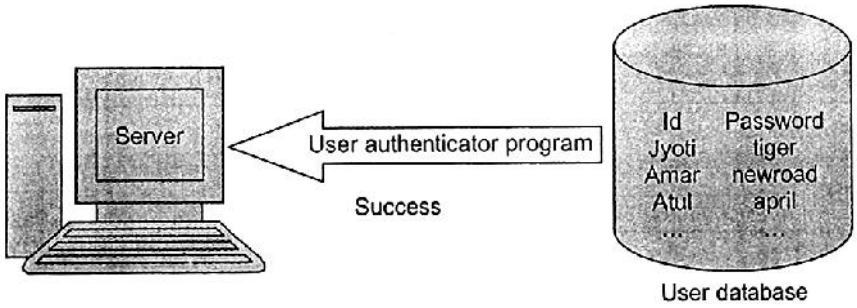
The server consults the user database to see if this user id and password combination exists there. This is a program that takes user id and password, checks it against the user database, and returns the result of the authentication.



User authenticator checks the user id and password against the user database.

Step 4: Authentication result:

Depending on the success or failure of the validation of the user id and the password, the user authenticator program returns an appropriate result back to the server.

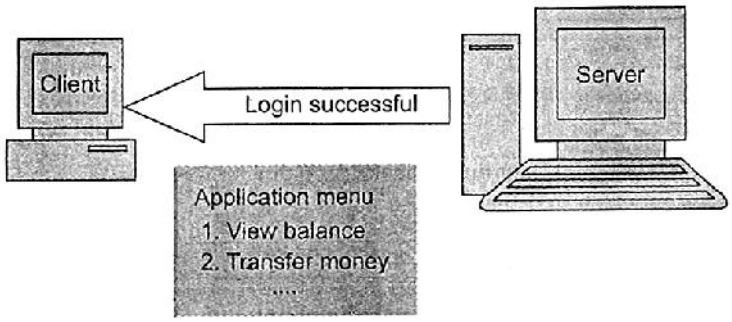


User authenticator program returns a success or failure message to the server.

Step 5: Inform user accordingly:

Depending on the outcome (success/failure), the server sends back an appropriate screen to the user. If the user authentication was

successful, the server typically sends a menu of options for the user, which lists the actions the user can perform. If the result of the user authentication was a failure, the server sends an error screen to the user.



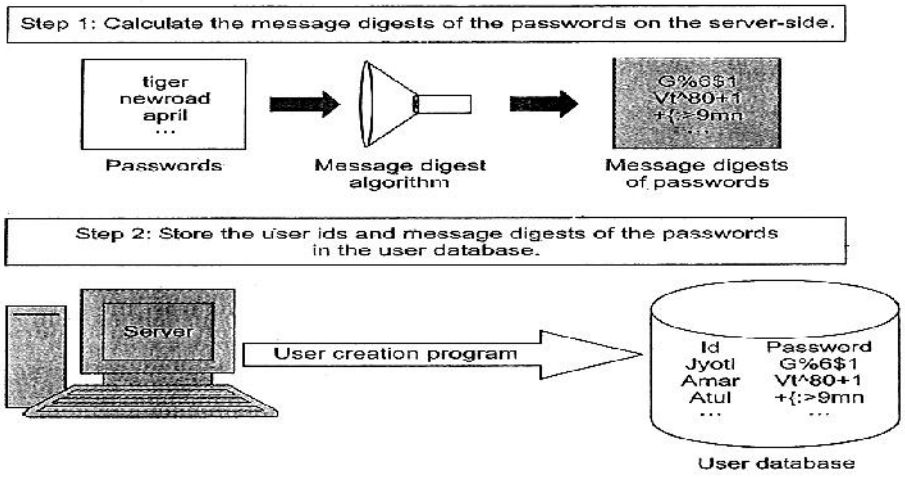
Server returns a success or failure result back to the user.

Message Digests of Passwords Mechanism:

A simple technology to avoid the storage and transmission of clear text passwords is the use of message digests.

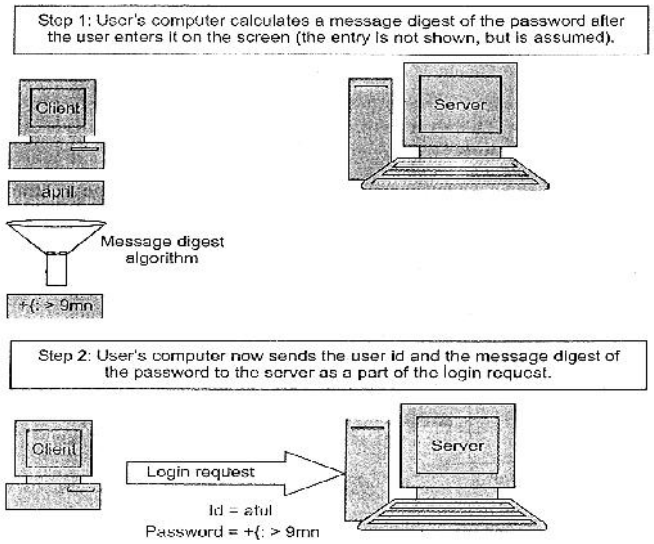
Step 1 (Storing message digests as derived passwords in the user database):

Rather than storing passwords. Storing message digests of the passwords in the user database.



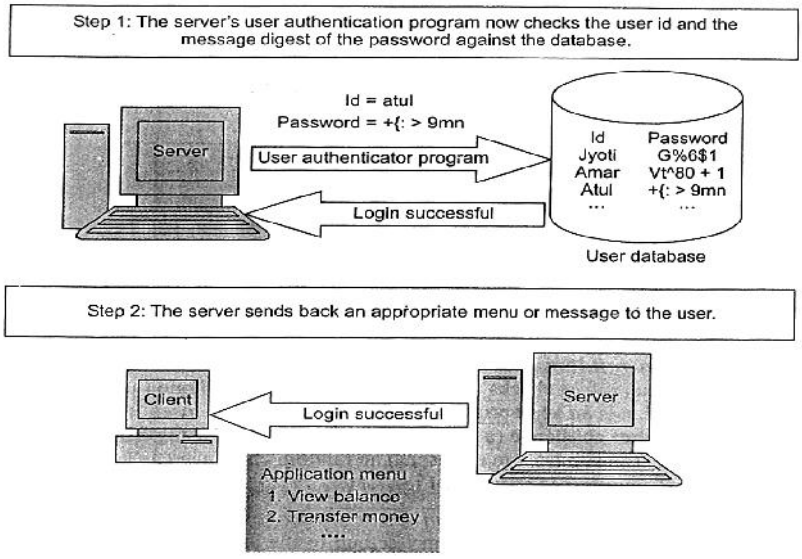
Step 2: User authentication:

When a user needs to be authenticated, the user enters the id and password, as usual. Now, the user's computer computes the message digest of the password and sends the user id and the message digest of the password to the server for authentication.



User authentication involving message digest of the password.

Step 3: Server-side validation



User authenticator program validates the user id and the message digest of the password.

The user id and the message digest of the password travel to the server over the communication link. The server passes these values to the user authentication program,which validates the user id and the message digest of the password against the database and returns an appropriate response back to the server.

8.12 SUMMARY

Intrusions are almost impossible to prevent. Hence, an attempt is made to detect them.

Intruders are classified into masquerader, misfeasor and clandestine user. A honeypot is a trap that attracts potential attackers.

8.13 UNIT END EXERCISES

1. Define virus. What are its phases.
2. Explain the Types/Categories of virus.
3. Write short note on Trojan Horse & Worms.
4. Write a short note on firewall.
5. Explain the different types of Firewalls - Packet Filters & Application Gateway.
6. What are the types of attack in Packet Filter?
7. Write a short note on Application Gateways.
8. List and explain all Firewall Configuration.
9. What are the limitations of Firewalls?
10. List & explain types of Intruders.
11. Define Honeypots.

8.14 REFERENCE FOR FURTHER READING

- Notes has been taken from : Introduction to Network Security by Atul Kahate.

*****