NEHRU COLLEGE OF ENGINEERING AND RESEARCH CENTRE
(NAAC Accredited)
(Approved by AICTE, Affiliated to APJ Abdul Kalam Technological University, Kerala)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

COURSE MATERIALS

CS 409 CRYPTOGRAPHY & NETWORK SECURITY

VISION OF THE INSTITUTION

To mould true citizens who are millennium leaders and catalysts of change through excellence in education.

MISSION OF THE INSTITUTION

NCERC is committed to transform itself into a center of excellence in Learning and Research in Engineering and Frontier Technology and to impart quality education to mould technically competent citizens with moral integrity, social commitment and ethical values.

We intend to facilitate our students to assimilate the latest technological know-how and to imbibe discipline, culture and spiritually, and to mould them in to technological giants, dedicated research scientists and intellectual leaders of the country who can spread the beams of light and happiness among the poor and the underprivileged.

ABOUT DEPARTMENT

- ♦ Established in: 2002
- ♦ Course offered : B.Tech in Computer Science and Engineering

  M.Tech in Computer Science and Engineering

  M.Tech in Cyber Security
- ♦ Approved by AICTE New Delhi and Accredited by NAAC
- ♦ Affiliated to the University of A P J Abdul Kalam Technological University.

DEPARTMENT VISION

Producing Highly Competent, Innovative and Ethical Computer Science and Engineering Professionals to facilitate continuous technological advancement.

DEPARTMENT MISSION

1. To Impart Quality Education by creative Teaching Learning Process
2. To Promote cutting-edge Research and Development Process to solve real world problems with emerging technologies.
3. To Inculcate Entrepreneurship Skills among Students.
4. To cultivate Moral and Ethical Values in their Profession.
5.

PROGRAMME EDUCATIONAL OBJECTIVES

PEO1: Graduates will be able to Work and Contribute in the domains of Computer Science and Engineering through lifelong learning.

PEO2: Graduates will be able to Analyse, design and development of novel Software Packages, Web Services, System Tools and Components as per needs and specifications.

PEO3: Graduates will be able to demonstrate their ability to adapt to a rapidly changing environment by learning and applying new technologies.

PEO4: Graduates will be able to adopt ethical attitudes, exhibit effective communication skills, Teamworkand leadership qualities.

PROGRAM OUTCOMES (POS)

Engineering Graduates will be able to:

1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one’s own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PROGRAM SPECIFIC OUTCOMES (PSO)

PSO1: Ability to Formulate and Simulate Innovative Ideas to provide software solutions for Real-time Problems and to investigate for its future scope.

PSO2: Ability to learn and apply various methodologies for facilitating development of high quality System Software Tools and Efficient Web Design Models with a focus on performance optimization.

PSO3: Ability to inculcate the Knowledge for developing Codes and integrating hardware/software products in the domains of Big Data Analytics, Web Applications and Mobile Apps to create innovative career path and for the socially relevant issues.

COURSE OUTCOMES

CO1	To summarize different classical encryption techniques
CO2	To identify mathematical concepts for different cryptographic algorithms.
CO3	To demonstrate cryptographic algorithms for encryption/key exchange
CO4	To summarize different authentication and digital signature schemes.
CO5	To identify security issues in network, transport and application layers and outline appropriate security protocols.
CO6	Describe the fundamentals of networks security, security architecture, threats and vulnerabilities

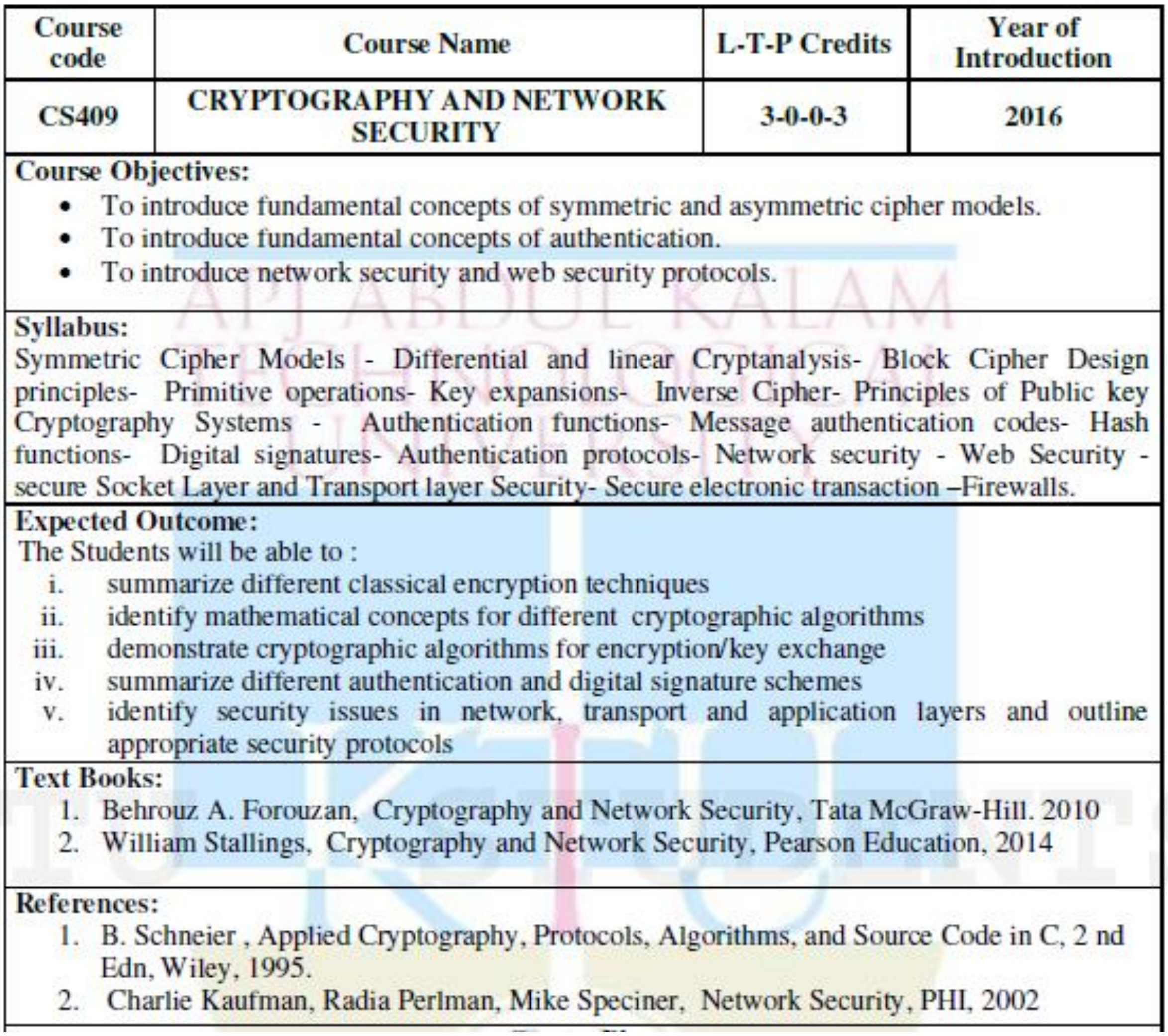
MAPPING OF COURSE OUTCOMES WITH PROGRAM OUTCOMES

	PO 1	**PO 2**	**PO 3**	**PO 4**	**PO 5**	**PO 6**	**PO 7**	**PO 8**	**PO 9**	**PO 10**	**PO 11**	**PO 12**
CO1	3	3										
CO2	3	3										
CO3		3		3				3				
CO4	2		2									
CO5	3	3										
CO6	2	2	2									

Note: H-Highly correlated=3, M-Medium correlated=2, L-Less correlated=1

PSO MAPPINGS ALSO NEEDS TO INCLUDE

SYLLABUS



Course code	Course Name	L-T-P Credits	Year of Introduction
CS409	CRYPTOGRAPHY AND NETWORK SECURITY	3-0-0-3	2016

Course Objectives:

- To introduce fundamental concepts of symmetric and asymmetric cipher models.
- To introduce fundamental concepts of authentication.
- To introduce network security and web security protocols.

Syllabus:

Symmetric Cipher Models - Differential and linear Cryptanalysis- Block Cipher Design principles- Primitive operations- Key expansions- Inverse Cipher- Principles of Public key Cryptography Systems - Authentication functions- Message authentication codes- Hash functions- Digital signatures- Authentication protocols- Network security - Web Security - secure Socket Layer and Transport layer Security- Secure electronic transaction –Firewalls.

Expected Outcome:

The Students will be able to :

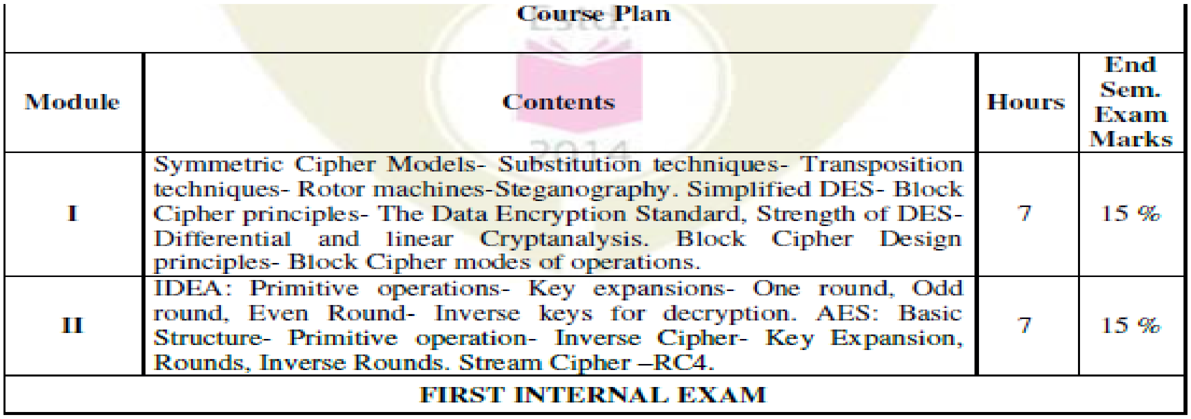
i. summarize different classical encryption techniques
ii. identify mathematical concepts for different cryptographic algorithms
iii. demonstrate cryptographic algorithms for encryption/key exchange
iv. summarize different authentication and digital signature schemes
v. identify security issues in network, transport and application layers and outline appropriate security protocols

Text Books:

1. Behrouz A. Forouzan, Cryptography and Network Security, Tata McGraw-Hill. 2010
2. William Stallings, Cryptography and Network Security, Pearson Education, 2014

References:

1. B. Schneier , Applied Cryptography, Protocols, Algorithms, and Source Code in C, 2 nd Edn, Wiley, 1995.
2. Charlie Kaufman, Radia Perlman, Mike Speciner, Network Security, PHI, 2002



Course Plan

Module	Contents	Hours	End Sem. Exam Marks
I	Symmetric Cipher Models- Substitution techniques- Transposition techniques- Rotor machines-Steganography. Simplified DES- Block Cipher principles- The Data Encryption Standard, Strength of DES-Differential and linear Cryptanalysis. Block Cipher Design principles- Block Cipher modes of operations.	7	15 %
II	IDEA: Primitive operations- Key expansions- One round, Odd round, Even Round- Inverse keys for decryption. AES: Basic Structure- Primitive operation- Inverse Cipher- Key Expansion, Rounds, Inverse Rounds. Stream Cipher –RC4.	7	15 %
FIRST INTERNAL EXAM			

III	Public key Cryptography: - Principles of Public key Cryptography Systems, Number theory- Fundamental Theorem of arithmetic, Fermat's Theorem, Euler's Theorem, Euler's Totient Function, Extended Euclid's Algorithm, Modular arithmetic. RSA algorithm- Key Management - Diffie-Hellman Key Exchange, Elliptic curve cryptography	7	15 %
IV	Authentication requirements- Authentication functions- Message authentication codes- Hash functions- SHA -1, MD5, Security of Hash functions and MACs- Authentication protocols-Digital signatures-Digital signature standards.	7	15 %
SECOND INTERNAL EXAM			
V	Network security: Electronic Mail Security: Pretty good privacy-S/MIME. IP Security: Architecture- authentication Header-Encapsulating Security payload- Combining Security associations-Key management.	7	20 %
VI	Web Security: Web Security considerations- secure Socket Layer and Transport layer Security- Secure electronic transaction. Firewalls-Packet filters- Application Level Gateway- Encrypted tunnels.	7	20 %
END SEMESTER EXAM			

Question Paper Pattern (End semester exam)

1. There will be ***FOUR*** parts in the question paper – **A, B, C, D**
2. **Part A**
    a. **Total marks : 40**
    b. ***TEN*** questions, each have **4 marks**, covering **all the SIX modules** (***THREE*** questions from **modules I & II**; ***THREE*** questions from **modules III & IV**; ***FOUR*** questions from **modules V & VI**). ***All*** questions have to be answered.
3. **Part B**
    a. **Total marks : 18**
    b. ***THREE*** questions, each having **9 marks**. One question is from **module I**; one question is from **module II**; one question ***uniformly*** covers **modules I & II**.
    c. ***Any TWO*** questions have to be answered.
    d. Each question can have ***maximum THREE*** subparts.
4. **Part C**
    a. **Total marks : 18**
    b. ***THREE*** questions, each having **9 marks**. One question is from **module III**; one question is from **module IV**; one question ***uniformly*** covers **modules III & IV**.
    c. ***Any TWO*** questions have to be answered.
    d. Each question can have ***maximum THREE*** subparts.
5. **Part D**
    a. **Total marks : 24**
    b. ***THREE*** questions, each having **12 marks**. One question is from **module V**; one question is from **module VI**; one question ***uniformly*** covers **modules V & VI**.
    c. ***Any TWO*** questions have to be answered.
    d. Each question can have ***maximum THREE*** subparts.
6. There will be ***AT LEAST* 60%** analytical/numerical questions in all possible combinations of question choices.

QUESTION BANK

MODULE I

Q:NO:	QUESTIONS	CO	KL	PAGE NO:
1	Explain Playfair cipher & Vernam cipher in detail.	CO1	K5	09
2	Convert “MEET ME” using Hill cipher with the key matrix Convert the cipher text back to plaintext.	CO1	K3	08
3	Explain simplified DES with example.	CO1	K5	15
4	Write short notes on rotor machines	CO1	K3	12
5	Steganography in detail	CO1	K2	11
6	Block cipher modes of operation	CO1	K2	30
7	Explain classical Encryption techniques in detail.	CO1	K5	05
8	Write short notes Security services	CO1	K2	04
9	Detail about Feistel cipher structure	CO1	K5	13
10	Explain Data Encryption Standard (DES) in detail.	CO1	K2	15

MODULE II

Q:NO:	QUESTIONS	CO	KL	PAGE NO:
1	How AES is used for encryption/decryption? Discuss with example.	CO2	K3	41
2	List the evaluation criteria defined by NIST for AES.	CO2	K3	21
3	Explain the IDEA primitive operations.	CO2	K2	35
4	Explain IDEA Odd and even rounds functions.	CO2	K5	38
5	Explain RC4 in detail	CO2	K2	52

MODULE III

1	State and explain the principles of public key cryptography.	CO3	K2	58
2	Explain Diffie Hellman key Exchange in detail with an example	CO3	K3	74
3	Explain the key management of public key encryption in detail	CO3	K2	71
4	Explain RSA algorithm in detail with an example	CO3	K2	66
5	Briefly explain the idea behind Elliptic Curve Cryptosystem.	CO3	K5	78
6	Perform encryption and decryption using RSA Alg. for the following. P=7; q=11; e=17; M=8.	CO3	K3	66
7	Discuss about Fermats Theorem and extended Euclidean.	CO3	K2	65
8	Explain Eulers and Totient theorem	CO3	K5	64
9	Explain Modular artematic	CO3	K5	66
MODULE IV				
1	Explain the classification of authentication function in detail	CO4	K5	82
2	Describe MD5 algorithm in detail. Compare its performance with SHA-1.	CO4	K2	90
3	Describe SHA-1 algorithm in detail. Compare its performance with MD5 and RIPEMD-160 and discuss its advantages.	CO4	K2	93
4	Describe RIPEMD-160 algorithm in detail. Compare its performance with MD5 and SHA-1.	CO4	K2	91
5	Describe HMAC algorithm in detail.	CO4	K2	89
6	Write and explain the Digital Signature Algorithm.	CO4	K6	101
7	Assume a client C wants to communicate with a server S using Kerberos	CO4	K4	98

MODULE V				
1	Explain the operational description of PGP.	CO5	K2	116
2	Write Short notes on S/MIME.	CO5	K2	116
3	Explain the architecture of IP Security.	CO5	K5	124
4	Write short notes on authentication header and ESP.	CO5	K6	134,141
5	Explain in detail the operation of Secure Socket Layer in detail.	CO5	K2	152
6	Explain Secure Electronic transaction with neat diagram.	CO5	K2	166
MODULE VI				
1	Different methods of web security threats	CO6	K3	150
2	Discuss about SSL architecture	CO6	K2	152
3	Explain connection process of SSL	CO6	K5	154
4	Describe SSL handshake protocol	CO6	K2	157
5	Discuss about TLS.	CO6	K2	163
6	Explain about SET and its features	CO6	K2	166
7	Discuss the working of SET Protocols	CO6	K2	170

APPENDIX 1

CONTENT BEYOND THE SYLLABUS

S:NO;	TOPIC	PAGE NO:
1	Functional encryption	175
2	Black-Box Impossibility Results	175
3	lattice-based cryptography and foundations	175

MODULE NOTES

Basics of Information and Network Security

- In daily life we use information for various purposes and use network for communication and exchange information between different parties.
- In many cases these information are sensitive so we need to take care that only authorized party can get that information.
- For maintaining such privacy we require some mechanism or physical device which ensures that it is safe. Such mechanism or physical devices are known as **security system**.
- **Computer Security:** The protection afforded to an automated information system in order to attain the applicable objectives of preserving the **integrity**, **availability**, and **confidentiality** of information system resources.
- This definition of computer security introduces three key objectives that are at the heart of computer security:
  1. **Confidentiality:** It covers two concepts

     Data Confidentiality: Assures that private or confidential information is not made available or disclosed to unauthorized individuals.

     Privacy: Assures that individuals control or influence what information related to them may be collected and stored and by whom and to whom that information may be disclosed.
  2. **Integrity:** It covers two concepts

     Data Integrity: Assures that information and programs are changed only in a specified and authorize manner.

     System Integrity: Assures that a system performs its intended function in an unimpaired manner, free from deliberate or inadvertent unauthorized manipulation of the system.
  3. **Availability:** Assures that systems work promptly and service is not denied to authorize user.
- **Unconditionally secure algorithm:** An algorithm or an encryption scheme is unconditionally secure if the attacker cannot obtain the corresponding plaintext from ciphertext no matter how much ciphertext is available.
- **Computationally secure algorithm:** An encryption scheme is said to be computationally secure if either of the following criteria is met:
  - The cost of breaking the cipher exceeds the value of the encrypted information.
  - The time required to break the cipher exceeds the useful lifetime of the information.
- **Threat:** A potential for violation of security, which exists when there is a circumstance, capability, action, or event that could breach security and cause harm. That is, a threat is a possible danger that might exploit vulnerability.

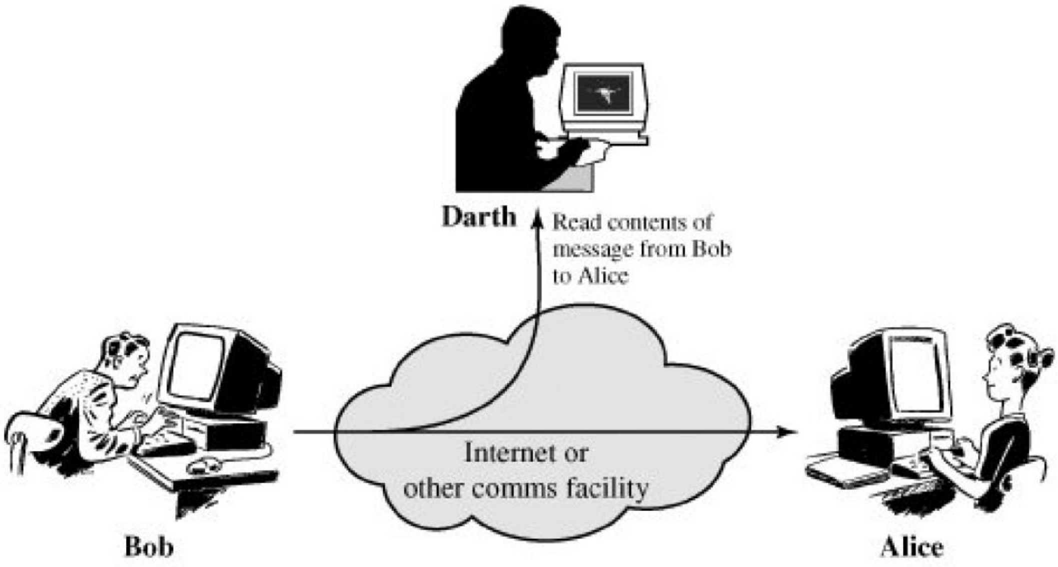
Security Attacks

- **Security Attacks:** An attack is an action that comprises the information or network security.
- There are two types of attacks:
  1. Passive Attack
  2. Active Attack

Passive Attack

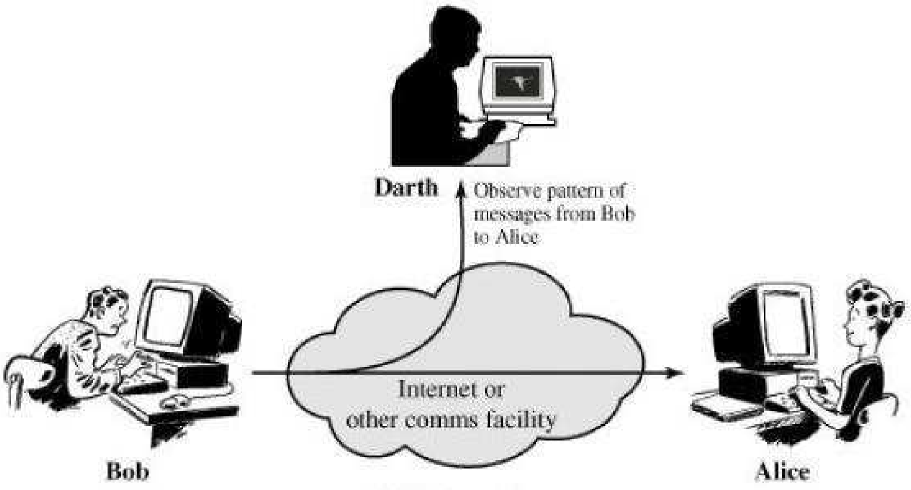
- **Passive Attack:** The attacker only monitors the traffic attacking the confidentiality of the data. It contains release of message contents and traffic analysis (in case of encrypted data).
  1. **Release of message contents:**
     - The release of message contents is easily understood.

- 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.


Darth
Read contents of message from Bob to Alice
Internet or other comms facility
Bob
Alice

(a) Release of message contents

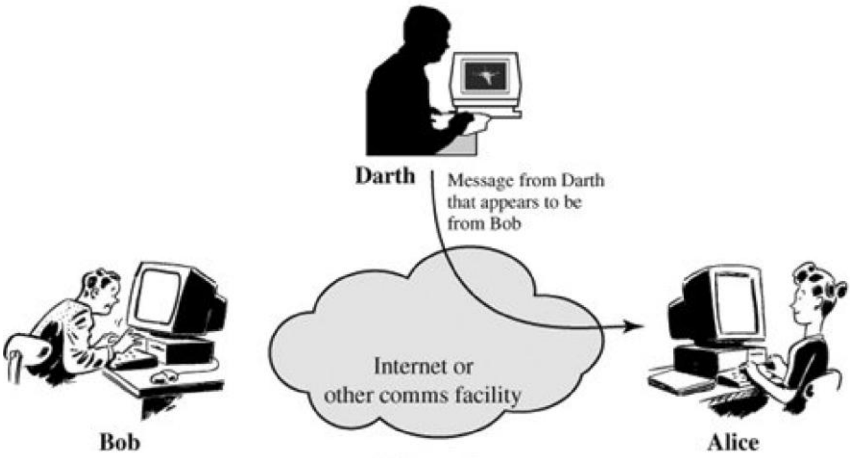
2. **Traffic analysis:**
   - A second type of passive attack, traffic analysis.
   - Suppose that we had a way of masking the contents of messages or other information.
   - Even if they captured the message, could not extract the information from the message.
   - The common technique for masking contents 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 send and received in an apparently normal fashion and the sender nor receiver is aware that a third party has read the messages or observed the traffic pattern.


Darth
Observe pattern of messages from Bob to Alice
Internet or other comms facility
Bob
Alice

(b) Traffic analysis

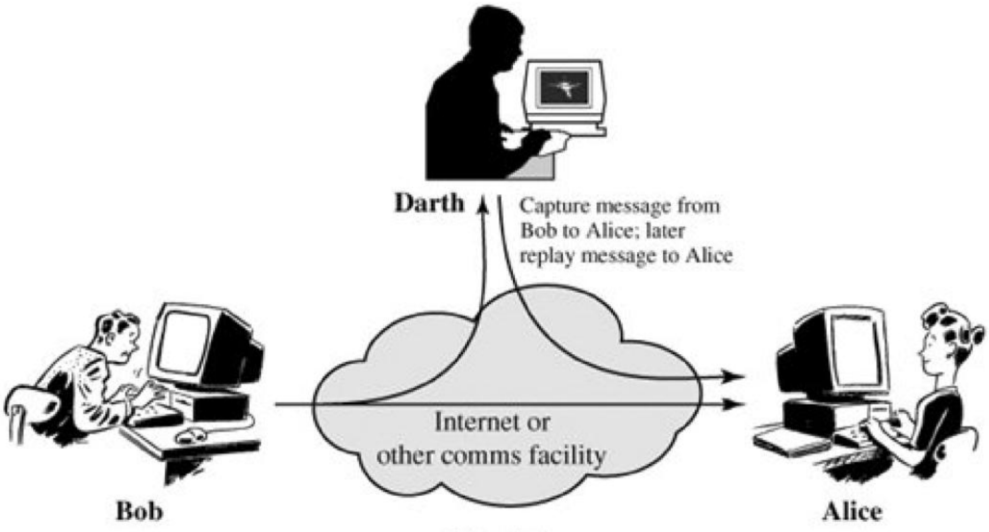
Active attack

- **Active attack:** Attacker tries to alter transmitted data. It includes masquerade, modification, replay and denial of service.
  1. **Masquerade:** A masquerade takes place when one entity pretends to be a different entity (Figure a). A masquerade attack usually includes one of the other forms of active attack.


Darth
Message from Darth that appears to be from Bob
Internet or other comms facility
Bob
Alice

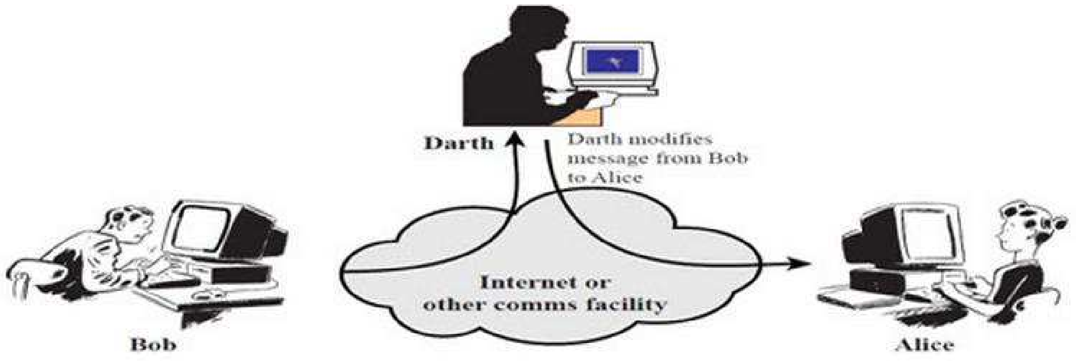
(a) Masquerade

  2. **Replay:** Replay involves the passive capture of a data unit and its subsequent retransmission to produce an unauthorized effect.


Darth
Capture message from Bob to Alice; later replay message to Alice
Internet or other comms facility
Bob
Alice

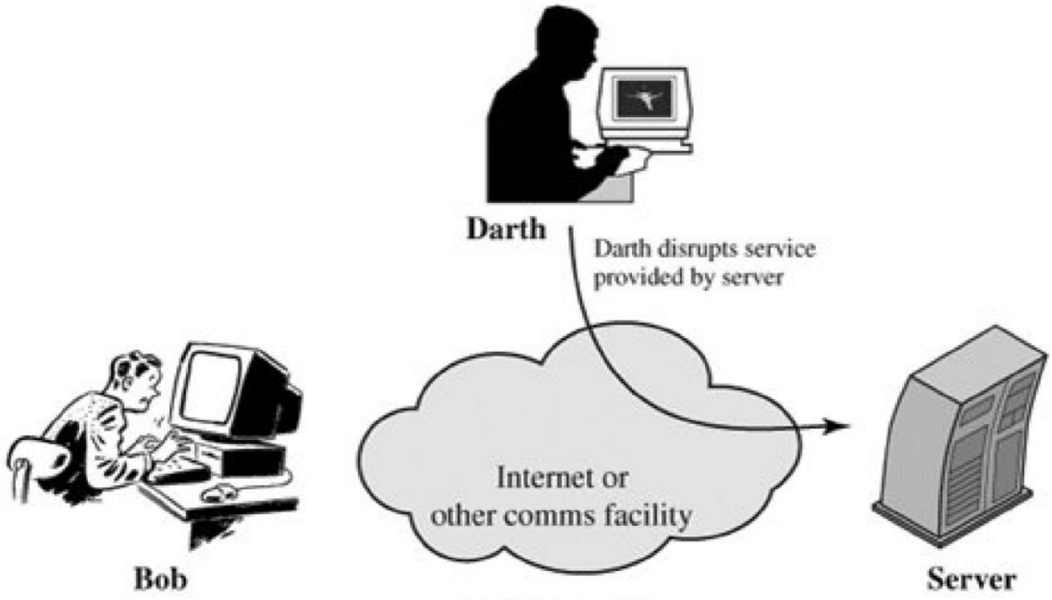
(b) Replay

  3. **Modification of messages:**
     - 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 c).
     - 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."


Darth
Darth modifies message from Bob to Alice
Internet or other comms facility
Bob
Alice

(c) Modification of messages

4. **Denial of service:**
   - The denial of service prevents or inhibits the normal use or management of communications facilities.
   - 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.

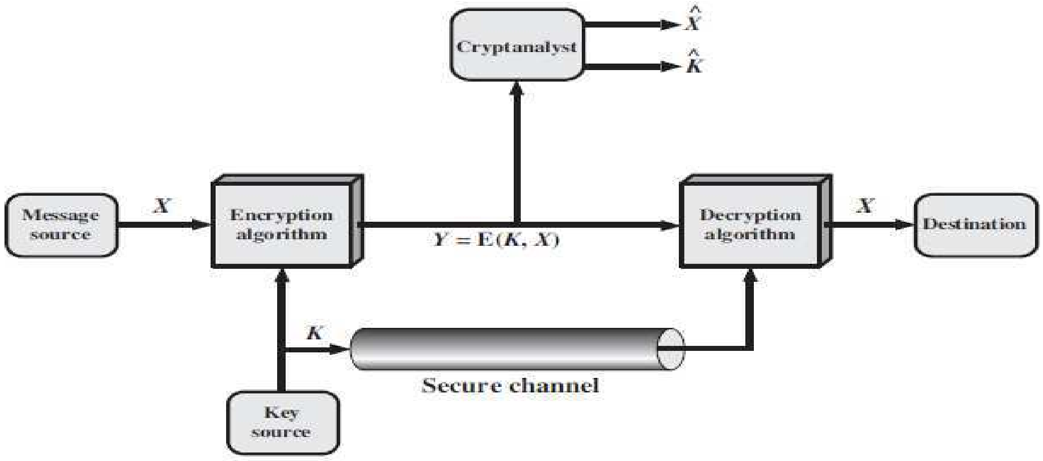

Darth
Darth disrupts service provided by server
Internet or other comms facility
Bob
Server

(d) Denial of service

Security services

- A security service is a processing or communicating service that can prevent or detect the above-mentioned attacks. Various security services are:
  - **Authentication:** the recipient should be able to identify the sender, and verify that the sender, who claims to be the sender, actually did send the message.
  - **Data Confidentiality:** An attacker should not be able to read the transmitted data or extract data in case of encrypted data. In short, confidentiality is the protection of transmitted data from passive attacks.
  - **Data Integrity:** Make sure that the message received was exactly the message the sender sent.
  - **Nonrepudiation:** The sender should not be able to deny sending the message. The receiver should not be able to deny receiving the message.

Symmetric Cipher Model



- A symmetric cipher model are broadly contains five parts.
- **Plaintext:** This is the original intelligible message.
- **Encryption algorithm:** The encryption algorithm performs various substitutions and transformations on the plaintext. It takes in plaintext and key and gives the ciphertext.
- **Secret key:** The key is a value independent of the plaintext and of the algorithm. Different keys will yield different outputs.
- **Ciphertext:** This is the scrambled message produced as output. It depends on the plaintext and the secret key.
- **Decryption algorithm:** Runs on the ciphertext and the key to produce the plaintext. This is essentially the encryption algorithm run in reverse.
- Two basic requirements of encryption are:
  1. Encryption algorithm should be strong. An attacker knowing the algorithm and having any number of ciphertext should not be able to decrypt the ciphertext or guess the key.
  2. The key shared by the sender and the receiver should be secret.
- Let the plaintext be $X = [X1, X2,\dots, X_M]$, key be $K = [K1, K2,\dots, K_J]$ and the ciphertext produced be $Y = [Y1, Y2,\dots, Y_N]$. Then, we can write
$$Y = E(K,X)$$
- Here E represents the encryption algorithm and is a function of plaintext X and key K.
- The receiver at the other ends decrypts the ciphertext using the key.
$$X = D(K,Y)$$
- Here D represents the decryption algorithm and it inverts the transformations of encryption algorithm.
- An opponent not having access to X or K may attempt to recover K or X or both.
- It is assumed that the opponent knows the encryption (E) and decryption (D) algorithms.
- If the opponent is interested in only this particular message, then the focus of the effort is to recover by generating a plaintext estimate $\widehat{X}$.
- If the opponent is interested in being able to read future messages as well then he will attempt to recover the key by making an estimate $\widehat{K}$.

Cryptography

- **Cryptography:** The area of study containing the principles and methods of transforming an intelligible message into one that is unintelligible, and then retransforming that message back to its original form.
- Cryptographic systems are characterized along three independent dimensions.
  1. **The types of operations used for transforming plaintext to ciphertext.** All encryption algorithms are based on two general principles substitution, and transposition. Basic requirement is that no information be lost. Most systems referred to as product system, involves multiple stages of substitutions and transpositions.
  2. **The number of keys used.** If both sender and receiver use the same key, the system is referred to as **symmetric, single-key, secret-key, or conventional encryption**. If the sender and receiver use different keys the system is referred to as **asymmetric, two-key, or public-key encryption**.
  3. **The way in which the plaintext is processed.** A block cipher process a block at a time and produce an output block for each input block. A stream cipher process the input element continuously, producing output one element at a time, as it goes along.

Cryptanalysis and Brute-Force Attack

- **Cryptanalysis:** Cryptanalytic attacks rely on the nature of the algorithm plus perhaps some knowledge of the general characteristics of the plaintext or even some simple plaintext-ciphertext pairs. This type of attack finds characteristics of the algorithm to find a specific plaintext or to find key.
- **Brute-force attack:** The attacker tries every possible key on a piece of ciphertext until plaintext is obtained. On average, half of all possible keys must be tried to achieve success.
- Based on the amount of information known to the cryptanalyst cryptanalytic attacks can be categorized as:
  - **Cipher text Only Attack:** The attacker knows only cipher text only. It is easiest to defend.
  - **Known plaintext Attack:** In this type of attack, the opponent has some plaintext-cipher text pairs. Or the analyst may know that certain plaintext patterns will appear in a message. For example, there may be a standardized header or banner to an electronic funds transfer message and the attacker can use that for generating plaintext-cipher text pairs.
  - **Chosen plaintext:** If the analyst is able somehow to get the source system to insert into the system a message chosen by the analyst, then a *chosen-plaintext* attack is possible. In such a case, the analyst will pick patterns that can be expected to reveal the structure of the key.
  - **Chosen Cipher text:** In this attack, the analyst has cipher text and some plaintext-cipher text pairs where cipher text has been chosen by the analyst.
  - **Chosen Text:** Here, the attacker has got cipher text, chosen plaintext-cipher text pairs and chosen cipher text-plaintext pairs.
- Chosen cipher text and chosen text attacks are rarely used.
- It is assumed that the attacker knows the encryption and decryption algorithms.
- Generally, an encryption algorithm is designed to withstand a known-plaintext attack.

Brute-force attack

- This type of attack becomes impractical as the key size increases as trying all the possible alternative keys for a very large key may take a huge amount of time.
- For example, for a binary key of 128 bits, 2^{128} keys are possible which would require around 5 X10^{24} years at the rate of 1 decryption per microsecond (current machine's speed).

- The Data Encryption Standard (DES) algorithm uses a 56-bit key a 128-bit key is used in AES.
- With massively parallel systems, even DES is also not secure against Brute Force attack.
- AES with its 128-bit key is secure since the time required to break it makes it impractical to try Brute-Force attack

Substitution Techniques

- Various conventional encryption schemes or substitution techniques are as under:

Caesar cipher

- The encryption rule is simple; replace each letter of the alphabet with the letter standing 3 places further down the alphabet.
- The alphabet is wrapped around so that Z follows A.
- Example:
  Plaintext: MEET ME AFTER THE PARTY
  Ciphertext: PHHW PH DIWHU WKH SDUWB
- Here, the key is 3. If different key is used, different substitution will be obtained.
- Mathematically, starting from a=0, b=1 and so on, Caesar cipher can be written as:

$$\mathrm{E}(p) = (p + k) \bmod (26)$$
$$\mathrm{D(C)} = (\mathrm{C} - k) \bmod (26)$$

- This cipher can be broken
  - If we know one plaintext-cipher text pair since the difference will be same.
  - By applying Brute Force attack as there are only 26 possible keys.

Monoalphabetic Substitution Cipher

- Instead of shifting alphabets by fixed amount as in Caesar cipher, any random permutation is assigned to the alphabets. This type of encryption is called monoalphabetic substitution cipher.
- For example, A is replaced by Q, B by D, C by T etc. then it will be comparatively stronger than Caesar cipher.
- The number of alternative keys possible now becomes 26!.
- Thus, Brute Force attack is impractical in this case.
- However, another attack is possible. Human languages are redundant i.e. certain characters are used more frequently than others. This fact can be exploited.
- In English ‘e’ is the most common letter followed by ‘t’, ‘r’, ‘n’, ’o’, ‘a’ etc. Letters like ‘q’, ‘x’, ‘j’ are less frequently used.
- Moreover, digrams like ‘th’ and trigrams like ‘the’ are also more frequent.
- Tables of frequency of these letters exist. These can be used to guess the plaintext if the plaintext is in uncompressed English language.

Playfair Cipher

- In this technique multiple (2) letters are encrypted at a time.
- This technique uses a 5 X 5 matrix which is also called key matrix.

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

- The plaintext is encrypted **two letters at a time**:
  - Break the plaintext into pairs of two consecutive letters.
  - If a pair is a repeated letter, insert a filler like ‘X‘in the plaintext, eg. "Balloon" is treated as "ba lx lo on".
  - If both letters fall in the same row of the key matrix, replace each with the letter to its right (wrapping back to start from end), eg. “AR" encrypts as "RM".
  - If both letters fall in the same column, replace each with the letter below it (again wrapping to top from bottom), eg. “MU" encrypts to "CM".
  - Otherwise each letter is replaced by the one in its row in the column of the other letter of the pair, eg. “HS" encrypts to "BP", and “EA" to "IM" or "JM" (as desired)
- Security is much improved over monoalphabetic as here two letters are encrypted at a time and hence there are 26 X 26 =676 diagrams and hence it needs a 676 entry frequency table.
- However, it can be broken even if a few hundred letters are known as much of plaintext structure is retained in cipher text.

Hill Cipher

- This cipher is based on linear algebra.
- Each letter is represented by numbers from 0 to 25 and calculations are done modulo 26.
- This encryption algorithm takes m successive plaintext letters and substitutes them with m cipher text letters.
- The substitution is determined by m linear equations. For *m* = 3, the system can be described as:

$$c_1 = (k_{11}p_1 + k_{12}p_2 + k_{13}p_3)\ mod\ 26$$
$$c_2 = (k_{21}p_1 + k_{22}p_2 + k_{23}p_3)\ mod\ 26$$
$$c_3 = (k_{31}p_1 + k_{32}p_2 + k_{33}p_3)\ mod\ 26$$

- This can also be expressed in terms of row vectors and matrices.

$$(c_1\ c_2\ c_3) = (p_1\ p_2\ p_3)\begin{pmatrix} k_{11} & k_{12} & k_{13} \\ k_{21} & k_{22} & k_{23} \\ k_{31} & k_{32} & k_{33} \end{pmatrix} mod\ 26$$

  Where **C** and **P** are row vectors of length 3 representing the plaintext and cipher text, and **K** is a 3 X 3 matrix representing the encryption key

- Key is an invertible matrix K modulo 26, of size m. For example:

$$K = \begin{pmatrix} 17 & 17 & 5 \\ 21 & 18 & 21 \\ 2 & 2 & 19 \end{pmatrix} \qquad K^{-1} = \begin{pmatrix} 4 & 19 & 15 \\ 15 & 17 & 6 \\ 24 & 0 & 17 \end{pmatrix}$$

- Encryption and decryption can be given by the following formulae:
  Encryption: $C = PK\ mod\ 26$
  Decryption: $P = CK^{-1}\ mod\ 26$

- The strength of the Hill cipher is that it completely hides single-letter frequencies.
- Although the Hill cipher is strong against a cipher text-only attack, it is easily broken with a known plaintext attack.
    - Collect m pair of plaintext-cipher text, where m is the size of the key.
    - Write the m plaintexts as the rows of a square matrix P of size m.
    - Write the m cipher texts as the rows of a square matrix C of size m.
    - We have that C=PK mod 26.
    - If P is invertible, then K=P^{-1}C mod 26,
    - If P is not invertible, then collect more plaintext-cipher text pairs until an invertible P is obtained.

The Vigenère cipher

- This is a type of polyalphabetic substitution cipher (includes multiple substitutions depending on the key). In this type of cipher, the key determines which particular substitution is to be used.
- To encrypt a message, a key is needed that is as long as the message. Usually, the key is a repeating keyword.
- For example, if the keyword is *deceptive*, the message "we are discovered save yourself" is encrypted as follows:

  Key: *deceptivedecept*

  Plaintext: wearediscovered

  Ciphertext: ZICVTWQNGRZGVTW
- Encryption can be done by looking in the Vigenere Table where ciphertext is the letter key's row and plaintext's column or by the following formula:

$$C_i = (P_i + K_{i\ mod\ m})\ mod\ 26$$

- Decryption is equally simple. The key letter again identifies the row. The position of the cipher text letter in that row determines the column, and the plaintext letter is at the top of that column.
- The strength of this cipher is that there are multiple ciphertext letters for each plaintext letter, one for each unique letter of the keyword.
- Thus, the letter frequency information is obscured however, not all knowledge of the plaintext structure is lost.

Vernam Cipher

- This system works on binary data (bits) rather than letters.
- The technique can be expressed as follows:

$$C_i = P_i \oplus K_i$$

  Where

  P_i = ith binary digit of plaintext.

  K_i = ith binary digit of key.

  C_i = ith binary digit of ciphertext.

  $\oplus$ = exclusive-or (XOR) operation
- Thus, the ciphertext is generated by performing the bitwise XOR of the plaintext and the key.
- Decryption simply involves the same bitwise operation:

$$P_i = C_i \oplus K_i$$

- The essence of this technique is the means of construction of the key.
- It was produced by the use of a running loop of tape that eventually repeated the key, so that in fact the system worked with a very long but repeating keyword.

- Although such a scheme has cryptanalytic difficulties, but it can be broken with a very long ciphertext or known plaintext as the key is repeated.

One-Time Pad

- In this scheme, a random key that is as long as the message is used.
- The key is used to encrypt and decrypt a single message, and then is discarded. Each new message requires a new key of the same length as the new message.
- This scheme is unbreakable.
- It produces random output that bears no statistical relationship to the plaintext.
- Because the ciphertext contains no information whatsoever about the plaintext, there is simply no way to break the code.
- For any plaintext of equal length to the ciphertext, there is a key that produces that plaintext.
- Therefore, if you did an exhaustive search of all possible keys, you would end up with many legible plaintexts, with no way of knowing which the intended plaintext was.
- Therefore, the code is unbreakable.
- The security of the one-time pad is entirely due to the randomness of the key.
- The one-time pad offers complete security but, in practice, has two fundamental difficulties:
  - There is the practical problem of making large quantities of random keys. Any heavily used system might require millions of random characters on a regular basis. Supplying truly random characters in this volume is a significant task.
  - Another problem is that of key distribution and protection. For every message to be sent, a key of equal length is needed by both sender and receiver.
- Because of these difficulties, the one-time pad is used where very high security is required.
- The one-time pad is the only cryptosystem that exhibits **perfect secrecy**.

Transposition Techniques

- A very different kind of mapping is achieved by performing some sort of permutation on the plaintext letters. This technique is referred to as a transposition cipher.
- The simplest such cipher is the **rail fence** technique.

Rail Fence Technique

- Encryption involves writing plaintext letters diagonally over a number of rows, then read off cipher row by row.
- For example, the text “meet me after the party” can be written (in 2 rows) as:

```
m   e   m   a   t   r   h   p   r   y
  e   t   e   f   e   t   e   o   a   t
```

- Ciphertext is read from the above row-by-row:
  MEMATRHPRYETEFETEAT
- This scheme is very easy to cryptanalyze as no key is involved.
- Transposition cipher can be made significantly more secure by performing more than one stage of transposition. The result is a more complex permutation that is not easily reconstructed.

Difference between Symmetric and Asymmetric key cryptography

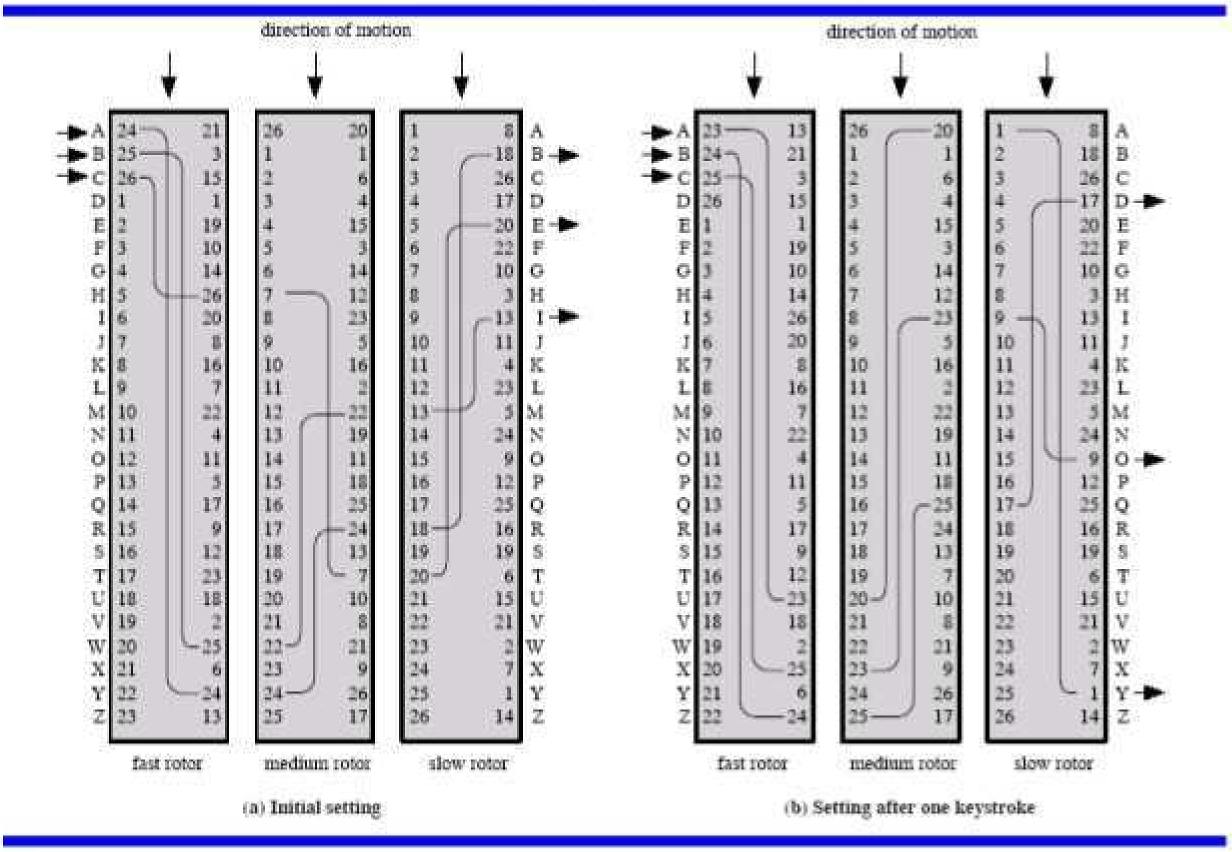
Symmetric key Cryptography	Asymmetric Key Cryptography
Symmetric key cryptography uses the same secret (private) key to encrypt and decrypt its data	Asymmetric key cryptography uses a public and a private key to encrypt and decrypt its data
The secret key must be known by both parties.	The public key is known to anyone with which they can encrypt the data but it can only be decoded by the person having the private key
In key distribution process, key information may have to be shared which decreases the security.	Here, the need for sharing key with key distribution center is eliminated.
Symmetric key encryption is faster than asymmetric key.	It is Slower than symmetric key encryption.
Basic operations used in encryption/ decryption are transposition and substitution.	It uses mathematical functions.

Steganography

- Plaintext message may be hidden in one of two ways.
    - Conceal the existence of the message-Steganography.
    - Render the message unintelligible to outsiders by various transformations of the text-Cryptography
- A simple but time consuming form of steganography is the one in which an arrangement of words or letters within an apparently normal text spells out the real message.
- For example, the sequence of first letters of each word of the overall message spells out the hidden message.
- Some other techniques that have been used historically are listed below:
    - **Character marking:** Selected letters of printed or typewritten text are overwritten in pencil. The marks are ordinarily not visible unless the paper is held at an angle to bright light.
    - **Invisible ink:** A number of substances can be used for writing but leave no visible trace until heat or some chemical is applied to the paper.
    - **Pin punctures:** Small pin punctures on selected letters are ordinarily not visible unless the paper is held up in front of a light.
    - **Typewriter correction ribbon:** Used between lines typed with a black ribbon, the results of typing with the correction tape are visible only under a strong light.
- Although these techniques may seem ancient, they have modern equivalents.
- For example, suppose an image has a resolution of 2048 X 3072 pixels where each pixel is denoted by 24 bits (Kodak CD photo format).
- The least significant bit of each 24-bit pixel can be changed without greatly affecting the quality of the image.
- The result is that you can hide a 2.3-megabyte message in a single digital snapshot.
- There are now a number of software packages available that take this type of approach to steganography.
- Steganography has a number of drawbacks when compared to encryption.
    - It requires a lot of overhead to hide a relatively few bits of information.
    - Once the system is discovered, it becomes virtually worthless.
- The advantage of steganography is that it can be employed by parties who have something to lose if the fact of their secret communication is discovered.

Rotor Machines

THREE-ROTOR MACHINES



(a) Initial setting

	fast rotor		medium rotor		slow rotor		
A	24	21	26	20	1	8	A
B	25	3	1	1	2	18	B
C	26	15	2	6	3	26	C
D	1	1	3	4	4	17	D
E	2	19	4	15	5	20	E
F	3	10	5	3	6	22	F
G	4	14	6	14	7	10	G
H	5	26	7	12	8	3	H
I	6	20	8	23	9	13	I
J	7	8	9	5	10	11	J
K	8	16	10	16	11	4	K
L	9	7	11	2	12	23	L
M	10	22	12	22	13	5	M
N	11	4	13	19	14	24	N
O	12	11	14	11	15	9	O
P	13	5	15	18	16	12	P
Q	14	17	16	25	17	25	Q
R	15	9	17	24	18	16	R
S	16	12	18	13	19	19	S
T	17	23	19	7	20	6	T
U	18	18	20	10	21	15	U
V	19	2	21	8	22	21	V
W	20	25	22	21	23	2	W
X	21	6	23	9	24	7	X
Y	22	24	24	26	25	1	Y
Z	23	13	25	17	26	14	Z

(b) Setting after one keystroke

	fast rotor		medium rotor		slow rotor		
A	23	13	26	20	1	8	A
B	24	21	1	1	2	18	B
C	25	3	2	6	3	26	C
D	26	15	3	4	4	17	D
E	1	1	4	15	5	20	E
F	2	19	5	3	6	22	F
G	3	10	6	14	7	10	G
H	4	14	7	12	8	3	H
I	5	26	8	23	9	13	I
J	6	20	9	5	10	11	J
K	7	8	10	16	11	4	K
L	8	16	11	2	12	23	L
M	9	7	12	22	13	5	M
N	10	22	13	19	14	24	N
O	11	4	14	11	15	9	O
P	12	11	15	18	16	12	P
Q	13	5	16	25	17	25	Q
R	14	17	17	24	18	16	R
S	15	9	18	13	19	19	S
T	16	12	19	7	20	6	T
U	17	23	20	10	21	15	U
V	18	18	21	8	22	21	V
W	19	2	22	21	23	2	W
X	20	25	23	9	24	7	X
Y	21	6	24	26	25	1	Y
Z	22	24	25	17	26	14	Z

- The basic principle of the rotor machine is illustrated in figure. The machine consists of a set of independently rotating cylinders through which electrical pulse can flow.
- Each cylinder has 26 input and 26 output pins, with internal wiring that connect each input pin to unique output pin.
- If we associate each input and output pin with a letter of the alphabet, then a single cylinder defines a monoalphabetic substitution.
- If we use multiple cylinders then we will obtain polyalphabetic substitution.

Block Cipher Principles

Stream Cipher and Block Cipher

- A **stream cipher** is one that encrypts a data stream one bit or one byte at a time. Example of stream cipher are the autokeyes vigenere cipher and vernam cipher.
- A **Block Cipher** is one in which a block of plaintext is treated as a whole and used to produce a cipher text block of equal length. Example of block cipher is DES.

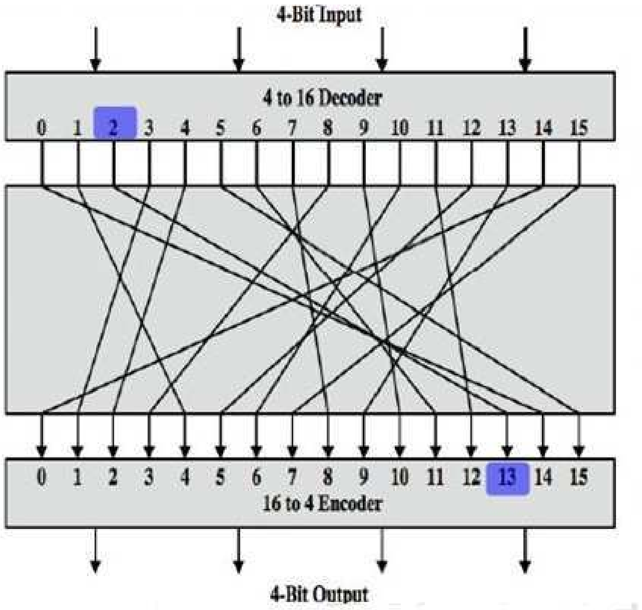
Motivation for the Feistel Cipher Structure

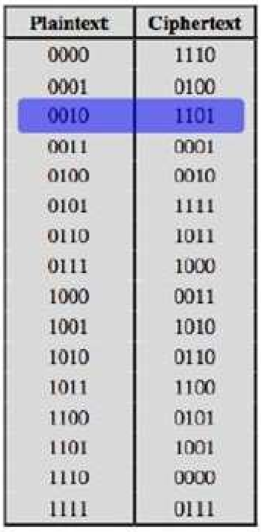
- A block cipher operates on a plaintext block of n bits to produce a ciphertext block of n bits.
- There are 2^n possible different plain text blocks and for the encryption to be reversible each must produce unique ciphertext block.
- Reversible encryption is also called as **singular encryption**. For example singular and non singular transformation for n=2.

Reversible Mapping	
Plaintext	**Ciphertext**
00	11
01	10
10	00
11	01

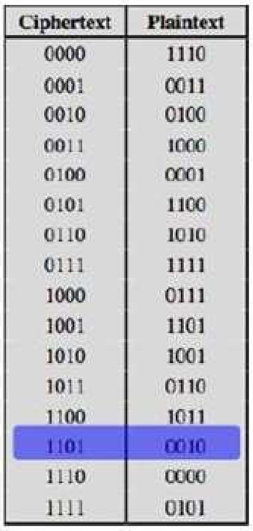
Irreversible Mapping	
Plaintext	**Ciphertext**
00	11
01	10
10	01
11	01

- If we limit ourselves to reversible mapping the number of different transformation is $2^n!$.
- Figure below illustrates the logic of a general substitution cipher for n=4





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



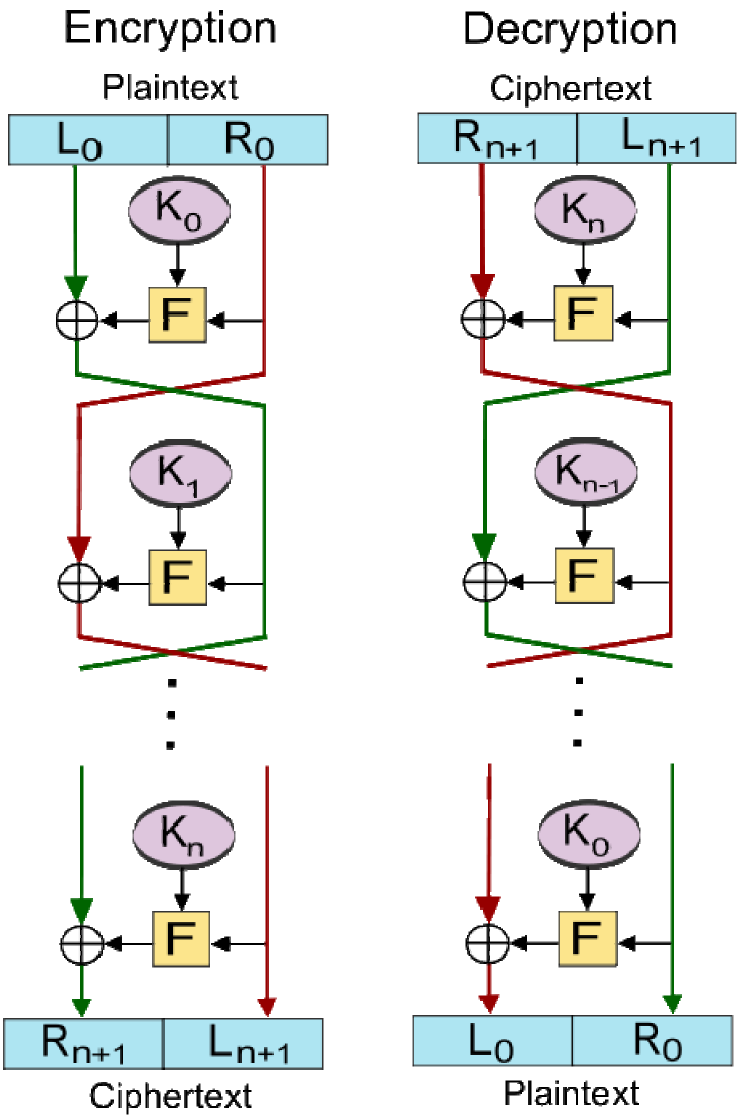
Ciphertext	Plaintext
0000	1110
0001	0011
0010	0100
0011	1000
0100	0001
0101	1100
0110	1010
0111	1111
1000	0111
1001	1101
1010	1001
1011	0110
1100	1011
1101	0010
1110	0000
1111	0101

- A 4-bit input produce one of 16 possible input states, which is mapped by substitution cipher into one of unique 16 possible output states, each of which is represented by 4-bit ciphertext.
- The encryption and decryption mapping can be defined by tabulation, as shown in table. This is the most general form of block cipher and can be used to define any reversible mapping between plaintext and ciphertext.
- Feistel refers to this as the ideal block cipher, because it allows for the maximum number of possible encryption mappings from the plaintext block.

- But there are practical problem with ideal block cipher is if we use small block size such as n=4 then it is vulnerable to statistical analysis of the plain text.
- If n is sufficiently large and an arbitrary reversible substitution between plaintext and ciphertext is allowed then statistical analysis is infeasible.
- Ideal block cipher is not practical for large block size according implementation and performance point of view.
- For such transformation mapping itself is a key and we require $nX2^n$ bits for n bit ideal block cipher which is not practical.
- In considering these difficulties, Feistel points out that what is needed is an approximation to the ideal cipher system for large n, built up out of components that are easily realizable.

The Feistel Cipher

- Feistel cipher is based on the idea that instead of using Ideal block cipher which degrades performance, a "substitution-permutation network" can be used.



Feistel Cipher Encryption

- The inputs to the encryption algorithm are a plaintext block of length b bits and a key K.

- The plaintext block is divided into two halves.
- The two halves of the data pass through rounds of processing and then combine to produce the ciphertext block.
- Each round has as inputs and derived from the previous round, as well as a subkey derived from the overall K.
- Any number of rounds could be implemented and all rounds have the same structure.
- A **substitution** is performed on the left half of the data. This is done by applying a round function F.
- The Round Function F: F takes right-half block of previous round and a subkey as input.
- The output of the function is XORed with the left half of the data.
- Left and right halves are then swapped.

Feistel Cipher Decryption

- The process of decryption with a Feistel cipher is same as the encryption process.
- The ciphertext is input to the algorithm and the subkeys are used in reverse order. That is, subkey of the last round in encryption is used in the first round in decryption, second last in the second round, and so on.

The exact realization of a Feistel network depends on the choice of the following parameters:

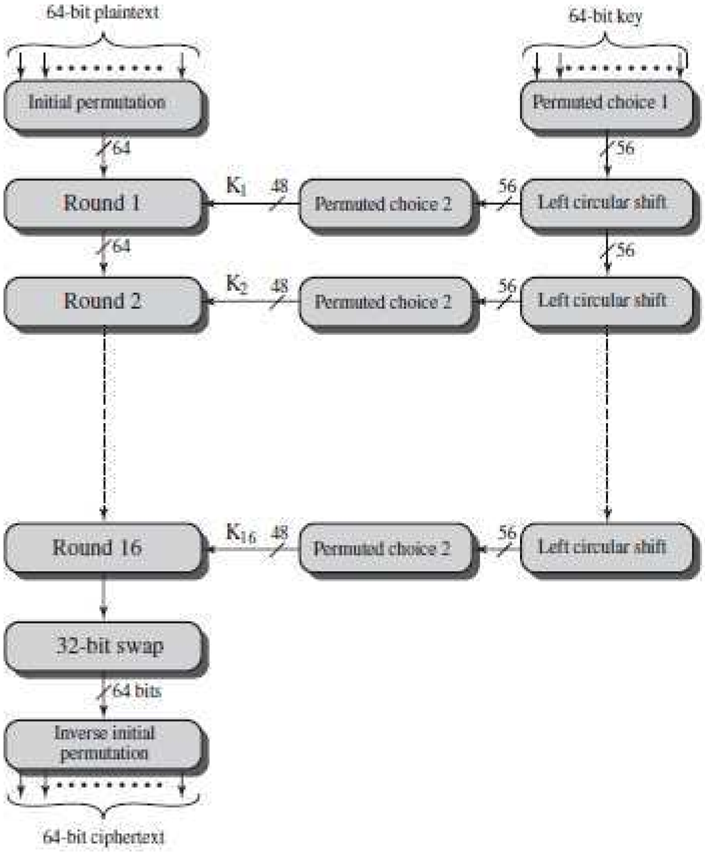
- **Block size**: Larger block sizes mean greater security but reduced encryption/decryption speed for a given algorithm. Traditionally, a block size of 64 bits is used which gives enough security without greatly affecting the speed.
- **Key size**: Larger key size means greater security but may decrease encryption/ decryption speed. The greater security is achieved by greater resistance to brute-force attacks and greater confusion. Key sizes of 64 bits or less are now widely considered to be inadequate, and 128 bits has become a common size.
- **Number of rounds**: The essence of the Feistel cipher is that a single round offers inadequate security but that multiple rounds offer increasing security. A typical size is 16 rounds.
- **Sub key generation algorithm**: Greater complexity in this algorithm leads to greater difficulty of cryptanalysis.
- **Round function F**: Again, greater complexity generally means greater resistance to cryptanalysis.
- There are two other considerations in the design of a Feistel cipher:
- **Fast software encryption/decryption**: In many cases, encryption is embedded in applications implementation (as software). Accordingly, the speed of execution of the algorithm becomes a concern.
- **Ease of analysis**: Although we would like to make our algorithm as difficult as possible to crypt analyze, there is great benefit in making the algorithm easy to analyze. That is, if The algorithm can be concisely and clearly explained, it is easier to analyze that algorithm for cryptanalytic vulnerabilities and therefore develop a high level of assurance as to its strength.

The Data Encryption Standard

- SDES encrypts 64-bit blocks using a 56-bit key and produces a 64-bit ciphertext.
- Same steps, with the same key, are used to reverse the encryption with the order of the keys reversed.
- The DES is widely used.

DES Encryption

- The DES encryption is shown in the figure below:



- Encryption function has two inputs: the plaintext to be encrypted and the key.
- The processing of the plaintext proceeds in three phases.
  - The 64-bit plaintext passes through an initial permutation (IP) that rearranges the bits to produce the permuted input.
  - The permuted output is then passed through sixteen rounds of the same function, which involves both permutation and substitution functions. The left and right halves from the last round are swapped to produce preoutput.
  - The preoutput is passed through a permutation that is the inverse of the initial permutation function, to produce the 64-bit cipher text.
- The right-hand portion of the figure shows the way in which the 56-bit key is used.
  - Initially, the key is passed through a permutation function.
  - Then, a sub key (k_i) is produces for each of the sixteen rounds by the combination of a left circular shift and a permutation.
  - The permutation function is the same for each round, but a different sub key is produced because of the repeated shifts of the key bits.

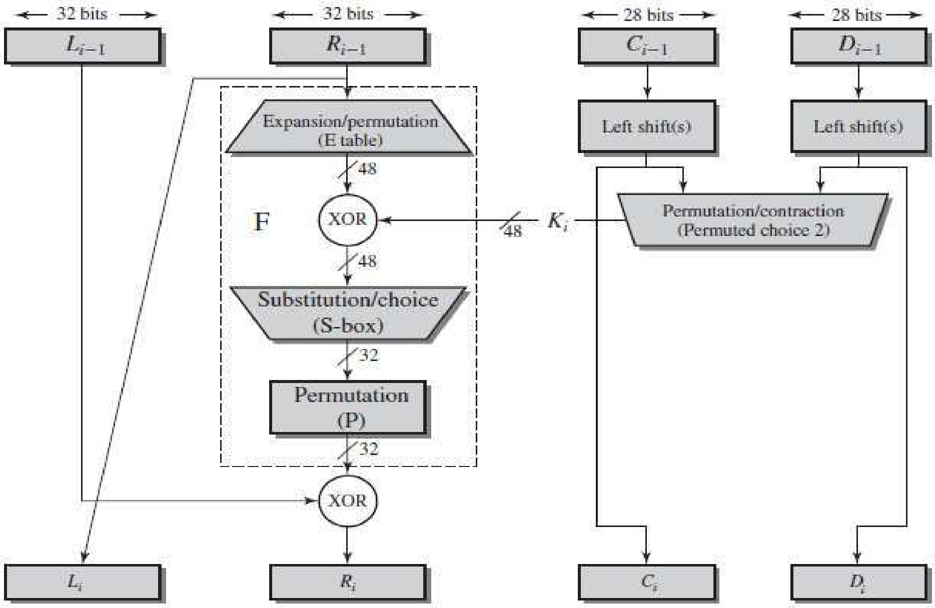
Initial Permutation (IP) and Inverse Initial Permutation (IP^{-1})

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

- The initial permutation and its inverse are defined by tables.
- The tables are to be interpreted as follows.
  - The input to a table consists of 64 bits numbered from 1 to 64.
  - The 64 entries in the permutation table contain a permutation of the numbers from 1 to 64.
  - Each entry in the permutation table indicates the position of a input bit in the output.
- Inverse permutation table nullifies the effect of initial permutation.

Details of Single Round

- Figure shows the internal structure of a single round.



- The left and right halves are treated as separate 32-bit quantities, labeled L (left) and R (right).
- The overall processing at each round can be summarized as:

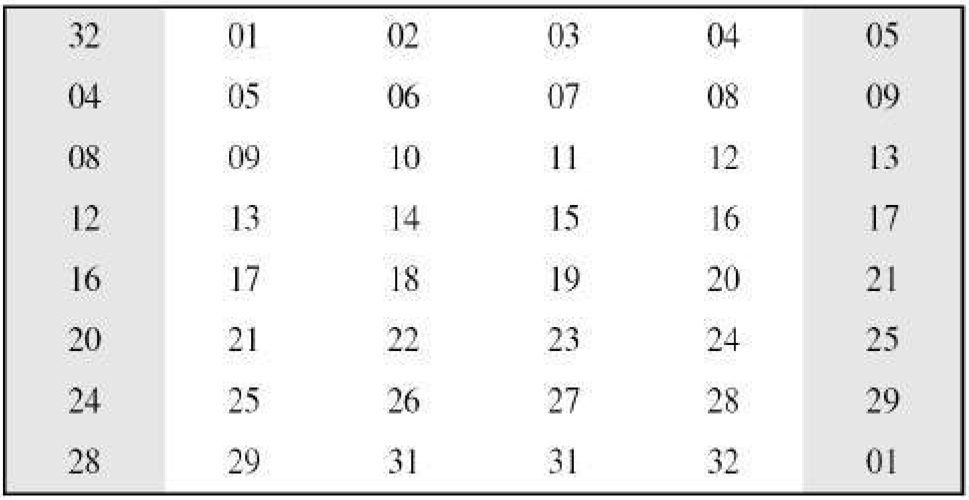
$$L_i = R_{i-1}$$
$$R_i = L_{i-1} \oplus F(R_{i-1}, K_i)$$

Expansion (E)

- The 32-bit input is first expanded to 48 bits.
  - Bits of input are split into groups of 4 bits.

- Each group is written as groups of 6 bits by taking the outer bits from the two adjacent groups. For example

... efgh ijkl mnop ... is expanded to
... defghi hijklm lmnopq ...

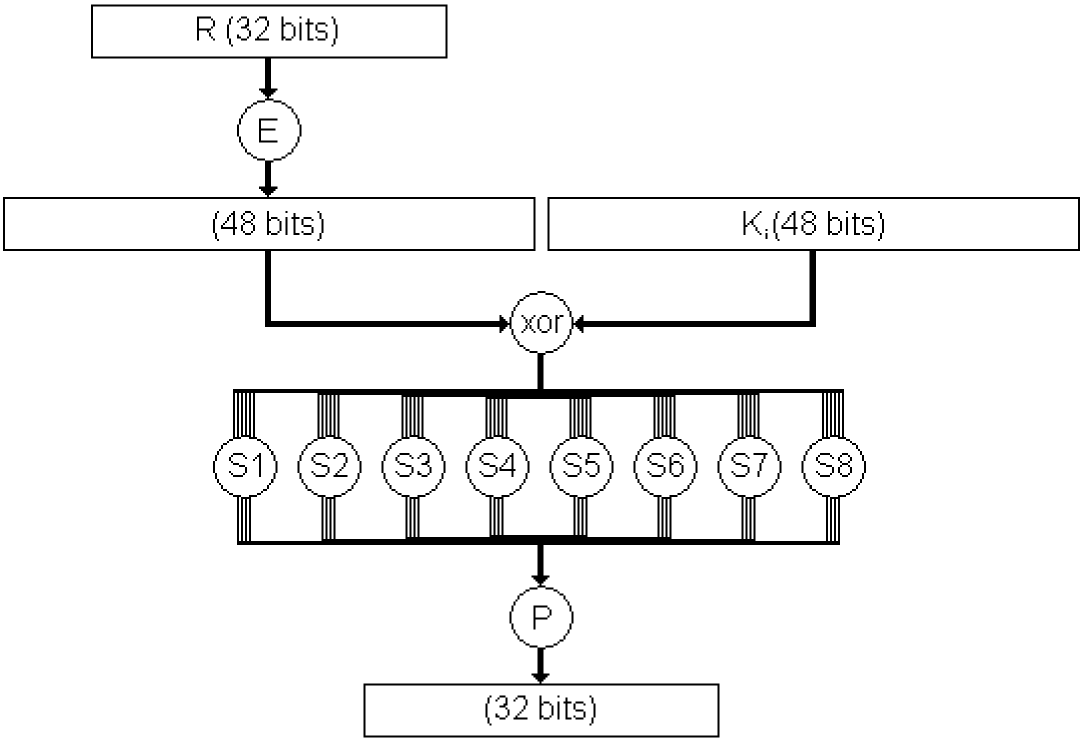


32	01	02	03	04	05
04	05	06	07	08	09
08	09	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	31	31	32	01

- The resulting 48 bits are XORed with K_i.

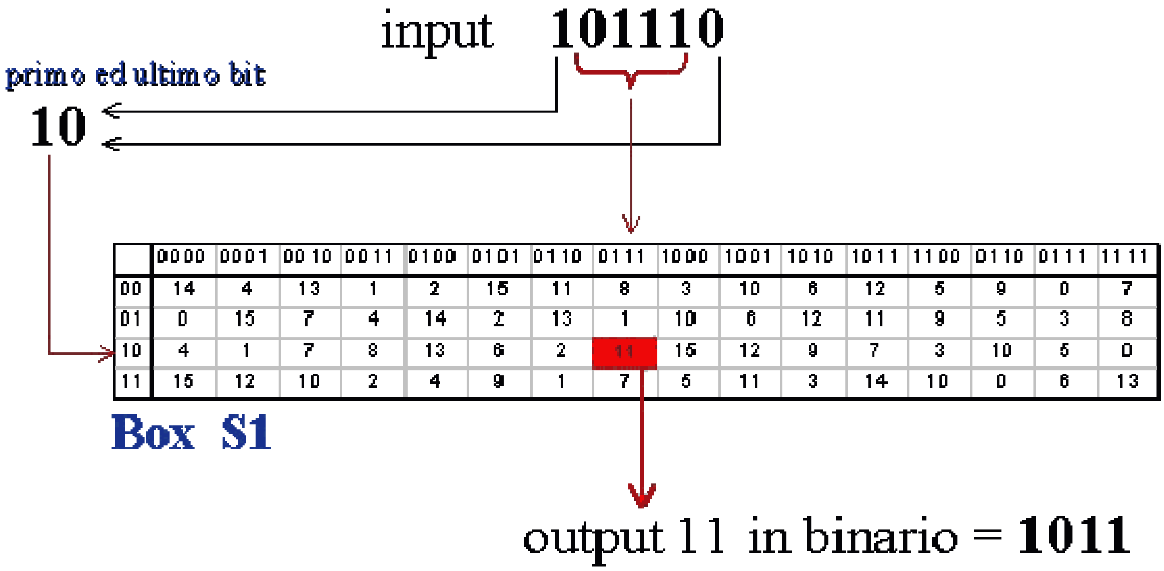
Substitution (S-Box)

- This 48-bit result is input to S-Boxes that perform a substitution on input and produces a 32-bit output.
- It is easy to understand S-Box by following figure.



- DES consists of a set of eight S-boxes.
- Each S-Box takes 6 bits as input and produces 4 bits as output.
- The first and last bits of the input to box form a 2-bit binary number which gives the binary value of row number.
- The middle four bits select one of the sixteen columns.

- The decimal value in the cell selected by the row and column is then converted to its 4-bit binary number to produce the output.
- For example, in S1, for input 101110, the row is 10 (row 2) and the column is 0111 (column 7).The value in row 2, column 7 is 11, so the output is 1011.



	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	0110	0111	1111
00	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
01	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
10	4	1	7	8	13	6	2	11	15	12	9	7	3	10	5	0
11	15	12	10	2	4	9	1	7	5	11	3	14	10	0	6	13

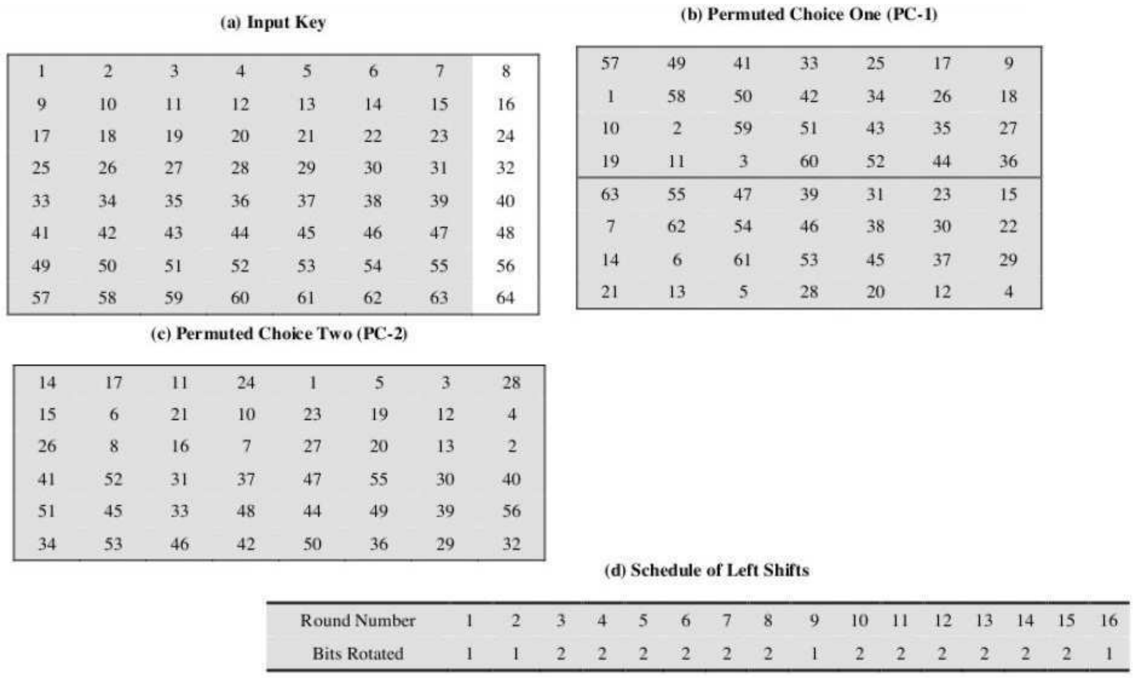
Permutation (P)

- The result is again permuted using a permutation table.

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

- A 64-bit key is used as input to the algorithm while only 56 bits are actually used. Every eighth bit is ignored. Sub-keys at each round are generated as given below:
    - The key is first permuted using a table named Permuted Choice One.
    - The resulting 56-bit key is divided into two 28-bit quantities, C_0 and D_0. At each round, C_{i-1} and D_{i-1} are separately subjected to a circular left shift of 1 or 2 bits, as governed by a table.
    - These shifted values are forwarded to the next round. They are also input to a permutation table-Permuted Choice Two.
    - The table produces a 48-bit output that serves as the round key k_i.



(a) Input Key

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

(b) Permuted Choice One (PC-1)

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

(c) Permuted Choice Two (PC-2)

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

(d) Schedule of Left Shifts

Round Number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Bits Rotated	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

DES Decryption

- Decryption in DES is same as encryption, except that the sub keys are used in reverse order.

Strength of DES

The Use of 56-Bit Keys

- DES has been developed from LUCIFER which used 128-bit keys.
- As a result, DES with only 56-bit key-length is considered insecure and devices have been proposed time and again showing that DES is no longer secure.

The Nature of the DES

- The only non-linear part of DES is the S-Boxes, design of which was not made public.
- If someone is able to find weakness in S-Box, then attack on DES is possible.
- Characteristics of the algorithm can be exploited as the algorithm is based on linear functions.

Algorithm Timing Attacks

- In this type of attack, the attacker exploits the fact that any algorithm takes different amount of time for different data.

A DES Example

- Let see example of DES and consider some of its implications. Although you are not expected to duplicate the example by hand, you will find it informative to study the hex patterns that occure from one step to the next.

Plaintext:	02468aceeca86420
Key:	0f1571c947d9e859
Ciphertext:	Da02ce3a89ecac3b

- **Result:** Above table shows plain text, key and cipher text when we apply all the steps of DES we will get cipher text as shown.

- **The Avalanche Effect:** A desirable property of any encryption algorithm is that a small change in either the plaintext or the key should produce a significant change in cipher text.
- In particular, a change in one bit of plaintext or one bit of the key should produce a change in many bits of the ciphertext. This is referred to as the avalanche effect.
- In DES 1 bit change in input will affect nearly 32 bit of output after all rounds.

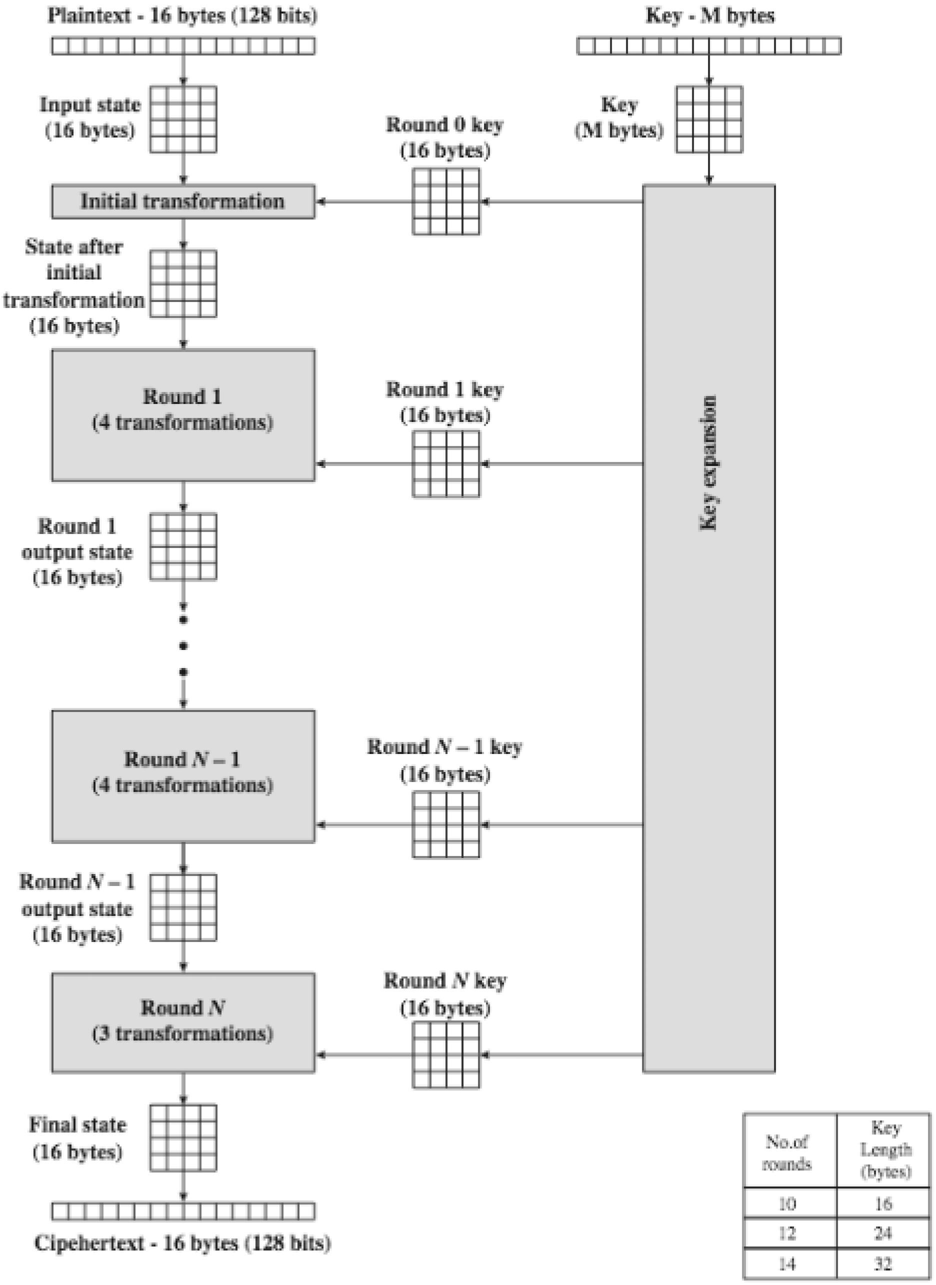
Block Cipher Design Principles

- The followed criteria need to be taken into account when designing a block cipher:
  - **Number of Rounds:** The greater the number of rounds, the more difficult it is to perform cryptanalysis, even for a weak function. The number of rounds is chosen so that efforts required to crypt analyze it becomes greater than a simple brute-force attack.
  - **Design of Function F**: F should be nonlinear and should satisfy strict avalanche criterion (SAC) and bit independence criterion (BIC).
  - **S-Box Design:** S-Box obviously should be non-linear and should satisfy SAC, BIC and Guaranteed Avalanche criteria. One more obvious characteristic of the S-box is its size. Larger S-Boxes provide good diffusion but also result in greater look-up tables. Hence, general size is 8 to 10.
  - **Key Generation Algorithm:** With any Feistel block cipher, the key is used to generate one sub key for each round. In general, sub keys should be selected such that it should be deduce sub keys from one another or main key from the sub key.

Advanced Encryption Standard (AES)

- The more popular and widely adopted symmetric encryption algorithm likely to be encountered nowadays is the Advanced Encryption Standard (AES). It is found at least six time faster than triple DES.
- A replacement for DES was needed as its key size was too small. With increasing computing power, it was considered vulnerable against exhaustive key search attack. Triple DES was designed to overcome this drawback but it was found slow.
- The features of AES are as follows
  - Symmetric key symmetric block cipher
  - 128-bit data, 128/192/256-bit keys
  - Stronger and faster than Triple-DES
  - Provide full specification and design details
  - Software implementable in C and Java

AES Structure

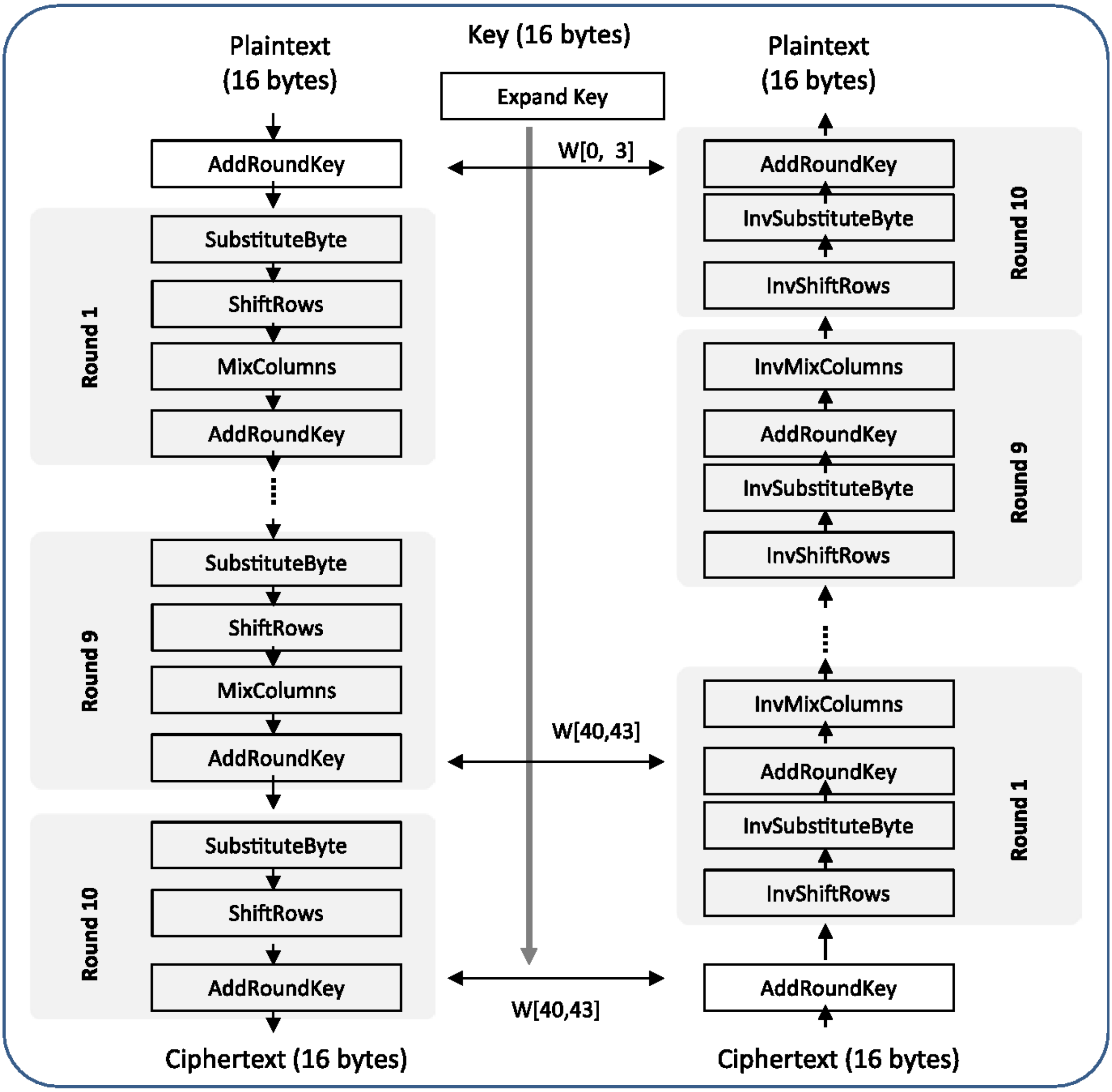

Plaintext - 16 bytes (128 bits)
Key - M bytes
Input state (16 bytes)
Round 0 key (16 bytes)
Key (M bytes)
Initial transformation
State after initial transformation (16 bytes)
Round 1 (4 transformations)
Round 1 key (16 bytes)
Key expansion
Round 1 output state (16 bytes)
Round N – 1 (4 transformations)
Round N – 1 key (16 bytes)
Round N – 1 output state (16 bytes)
Round N (3 transformations)
Round N key (16 bytes)
Final state (16 bytes)
Cipehertext - 16 bytes (128 bits)

No.of rounds	Key Length (bytes)
10	16
12	24
14	32

- Figure shows the overall structure of the AES encryption process.
- Plain text block size is 128 bits (16 bytes).
- Key size is depends on number of round 128, 192, or 256 bit as shown in table.
- Based on key size AES is named as AES-128, AES-192, or AES-256.
- The input 128 bit block, this block is arranged in the form of 4 X 4 square matrix of bytes. This block is copied into the **state** array, which is modified at each stage of encryption or decryption. After the final stage, State is copied to an output matrix.
- There is a initial single transformation (AddRoundKey) before the first round which can be considered Round 0.
- The first N-1 rounds consist of four distinct transformation function: SubBytes, ShiftRows, MixColumns, and AddRoundKey,which are described subsequently.
- The final round contains only first three transformations of above round.

- Each transformation takes one or more 4 X 4 matrices as input and produces a 4 X 4 matrix as output.
- The key expansion function generates N+1 round key each of which is distinct 4 X 4 matrices. Each round key serves as one of the inputs to the AddRoundKey transformation in each round.

Detail Structure



- Figure shows detail encryption Decryption process of AES.
- Lets discuss Several comments about AES structure:
  1. It is not a Feistel structure. As we know in feistel structure half of the data block is used to modify the other half of the data block and then the halves are swapped. While in AES we use full data block as a single matrix during each round.
  2. The key is expanded into an array of fourty-four 32-bit words. And such four word (128-bit) serves as round key for each round.
  3. Four different stages are used one of permutation and three of substitution:
     - **SubBytes**: Uses an S-box to perform a byte-by-byte substitution of the block.
     - **ShiftRows**: A simple permutation.

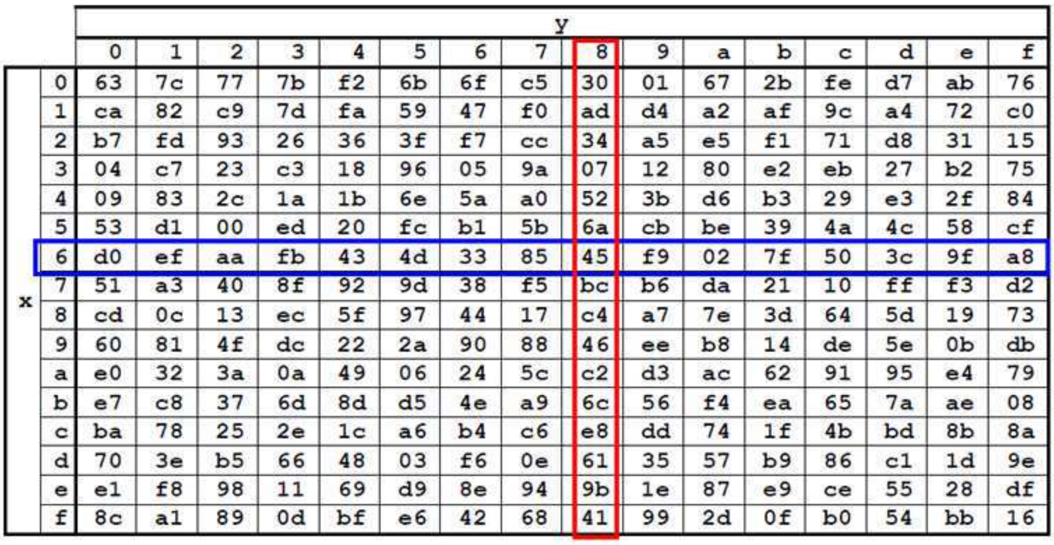
- **MixColumns**: A substitution that makes use of arithmetic over bytes.
- **AddRoundKey**: A simple bitwise XOR of the current block with a portion of the expanded key.

4. The structure is quite simple for both encryption and decryption it begins with AddRoundKey, followed by nine rounds of all four stages, followed by tenth round of three stages.
5. Only AddRoundKey stage use key for this reason, the cipher begins and ends with an AddRoundKey stage. Any other stage, applied at the beginning or end, is reversible without knowledge of the key and so would add no security.
6. The AddRoundKey stage is in effect, a form of Vernam cipher and by itself would not be formidable. The other three stages together provide confusion, diffusion, and nonlinearity, but by themselves would provide no security because they do not use the key.
7. Each stage is easily reversible.
8. In AES decryption algorithm is not identical to encryption algorithm.
9. Once it is established that all four stages are reversible, it is easy to verify that decryption does recover the plain text.
10. For making AES reversible the final round of both encryption and decryption are consists of only three stages.

AES Transformation Function

Substitute bytes Transformation (Forward & Inverse)

- Substitute bytes transformation is simple table lookup. There is separate table for forward and inverse operation.
- 16 X 16 matrix of byte value called s-box that contains the permutation of all 256 8-bit values. Each individual byte of state is mapped into a new byte in the following way.
- The left most 4-bit of the byte are used as row number and right most 4-bit are used as column number. Now row and column number serves as index into the s-box to select unique 8-bit output value.



		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

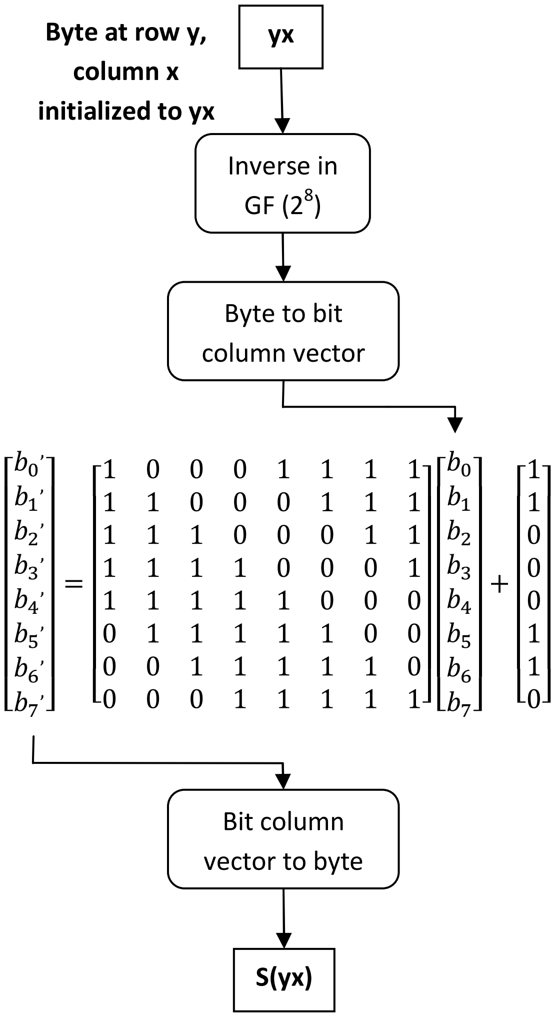
AES S-Box

- For example hexadecimal value 68 is referred to row 6 and column 8 and value in table at that position is 45 so byte value 68 is replaced with 45.
- For inverse substitute byte procedure is same but S-box is different. Reverse of above example is shown in figure.

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
	1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
	2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
	3	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
	4	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
	5	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
	6	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
	7	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
	8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
	9	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
	a	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
	b	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
	c	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
	d	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
	e	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
	f	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

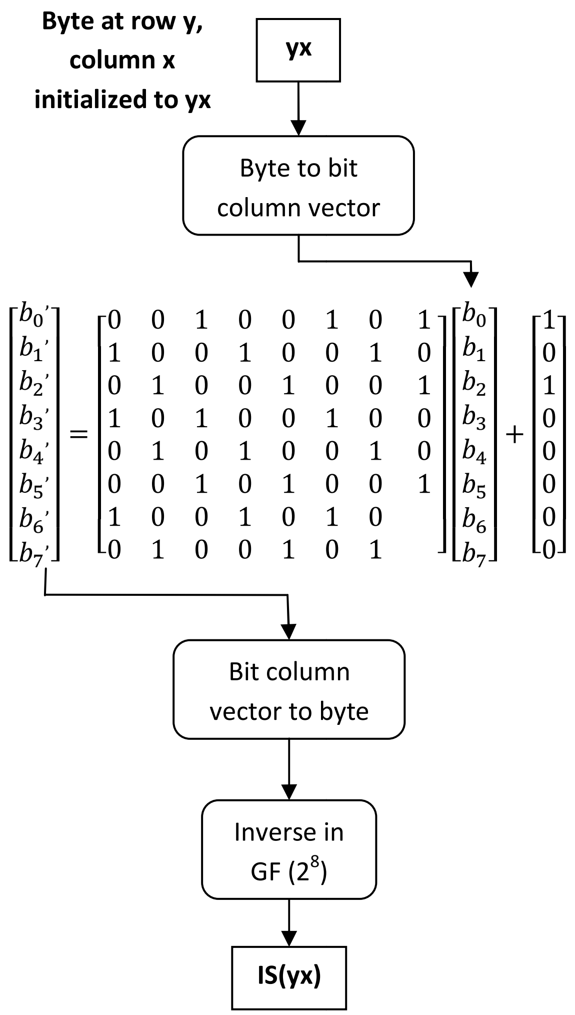
AES Inverse S-Box

- S-box is constructed in the following fashion:



$$\begin{bmatrix} b_0' \\ b_1' \\ b_2' \\ b_3' \\ b_4' \\ b_5' \\ b_6' \\ b_7' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

(a) Calculation of byte at row y, column x of S-box

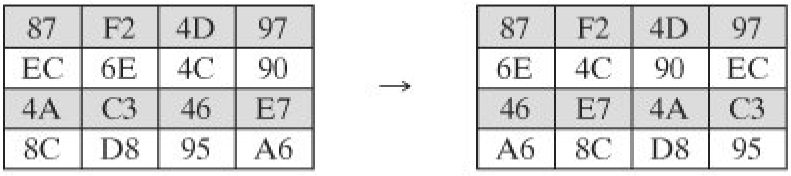


$$\begin{bmatrix} b_0' \\ b_1' \\ b_2' \\ b_3' \\ b_4' \\ b_5' \\ b_6' \\ b_7' \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

(b) Calculation of byte at row y, column x of IS-

- Construction of S-box:
  1. Initialize the S-box with the byte values in ascending sequence row by row.
  2. Map each byte in the S-box to its multiplicative inverse in the finite field GF (2^8). The value {00} is mapped to itself.
  3. Consider that each byte in the S-box consist of 8 bits labeled (b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0). Apply the transformation using matrix multiplication as shown in figure.
  4. Finally convert that bit column vector to byte.
- Construction of IS-box:
  1. Initialize the IS-box with the byte values in ascending sequence row by row.
  2. Consider that each byte in the IS-box consist of 8 bits labeled (b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0). Apply the transformation using matrix multiplication as shown in figure.
  3. Convert that bit column vector to byte.
  4. Map each byte in the IS-box to its multiplicative inverse in the finite field GF (2^8).

ShiftRows Transformation (Forward & Inverse)



87	F2	4D	97
EC	6E	4C	90
4A	C3	46	E7
8C	D8	95	A6

→

87	F2	4D	97
6E	4C	90	EC
46	E7	4A	C3
A6	8C	D8	95

- The **forward shift row transformation** is performed as below:
  1. The first row of state is not altered.
  2. In second row we apply 1-byte circular left shift.
  3. In third row we apply 2-byte circular left shift.
  4. In fourth row we apply 3-byte circular left shift.
- In the **inverse shift row transformation** we apply right circular rotation.
  1. The first row of state is not altered.
  2. In second row we apply 1-byte circular right shift.
  3. In third row we apply 2-byte circular right shift.
  4. In fourth row we apply 3-byte circular right shift.

MixColumns Transformation (Forward & Inverse)

- In the forward MixColumn transformation each byte of a column is mapped into a new value that is a function of all bytes in that column.
- The transformation can be defined by the following matrix multiplication on state:

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix}\begin{bmatrix} S_{0,0} & S_{0,1} & S_{0,2} & S_{0,3} \\ S_{1,0} & S_{1,1} & S_{1,2} & S_{1,3} \\ S_{2,0} & S_{2,1} & S_{2,2} & S_{2,3} \\ S_{3,0} & S_{3,1} & S_{3,2} & S_{3,3} \end{bmatrix} = \begin{bmatrix} S_{0,0}' & S_{0,1}' & S_{0,2}' & S_{0,3}' \\ S_{1,0}' & S_{1,1}' & S_{1,2}' & S_{1,3}' \\ S_{2,0}' & S_{2,1}' & S_{2,2}' & S_{2,3}' \\ S_{3,0}' & S_{3,1}' & S_{3,2}' & S_{3,3}' \end{bmatrix}$$

- In this case, the individual additions and multiplications are performed in GF (2^8).
- In the inverse MixColumn transformation procedure is same but matrix is different which is shown below.

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix}\begin{bmatrix} S_{0,0} & S_{0,1} & S_{0,2} & S_{0,3} \\ S_{1,0} & S_{1,1} & S_{1,2} & S_{1,3} \\ S_{2,0} & S_{2,1} & S_{2,2} & S_{2,3} \\ S_{3,0} & S_{3,1} & S_{3,2} & S_{3,3} \end{bmatrix} = \begin{bmatrix} S_{0,0}' & S_{0,1}' & S_{0,2}' & S_{0,3}' \\ S_{1,0}' & S_{1,1}' & S_{1,2}' & S_{1,3}' \\ S_{2,0}' & S_{2,1}' & S_{2,2}' & S_{2,3}' \\ S_{3,0}' & S_{3,1}' & S_{3,2}' & S_{3,3}' \end{bmatrix}$$

AddRoundKey Transformation

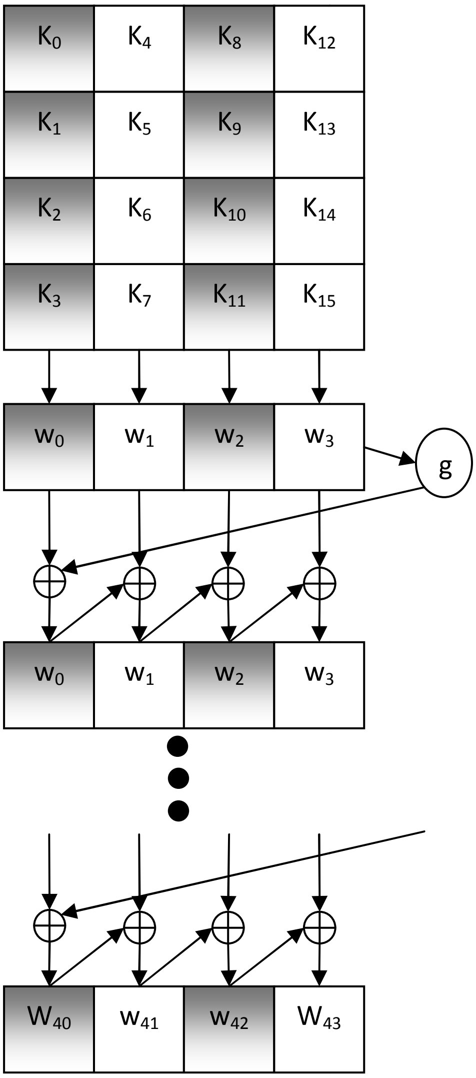
- In this transformation 128 bits state are bitwise XORed with the 128 bits of the round key.
- It is viewed as a byte level operation.
- Example

$$\begin{pmatrix} 47 & 40 & A3 & 4C \\ 37 & D4 & 70 & 9F \\ 94 & E4 & 3A & 42 \\ ED & A5 & A6 & BC \end{pmatrix} \oplus \begin{pmatrix} AC & 19 & 28 & 57 \\ 77 & FA & D1 & 5C \\ 66 & DC & 29 & 00 \\ F3 & 21 & 41 & 6A \end{pmatrix} = \begin{pmatrix} EB & 59 & 8B & 1B \\ 40 & 2E & A1 & C3 \\ F2 & 38 & 13 & 42 \\ 1E & 84 & E7 & D6 \end{pmatrix}$$

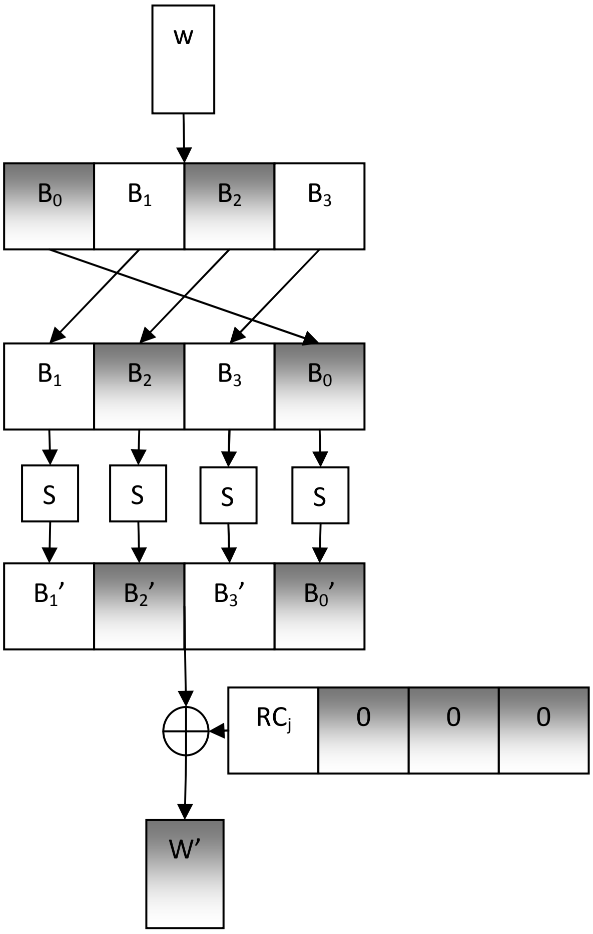
- Inverse of AddRoundKey is same because inverse of XOR is again XOR.

AES Key Expansion

- AES takes 16-byte key as input.
- As shown in figure below key expansion process is straight forward.



(a) Overall Key expansion algorithm.



(b) Function g.

- First of all key is stores in 4X4 matrix in column major matrix as shown in figure.
- Each column combines to form 32 bit word.
- Than we apply function g to every fourth word that is w3, w7, w11 etc.

- Than X-OR operation is performed as shown in figure to obtain next four word. And this process continues till generation of all words.
- As shown in figure (b) internal structure of function g.
- First we convert word to 4 byte.
- Then apply circular left shift operation.
- Then apply substitute byte operation using S-box which is same as S-box of AES encryption process.
- Than we apply X-OR operation with round constant which have least significant 3 byte as zero and most significant byte is depend on round number which is shown in table below.

Round (j)	1	2	3	4	5	6	7	8	9	10
RC[j]	01	02	04	08	10	20	40	80	1B	36

- And output of this function is used for X-OR operation as shown in figure (a).

AES Example

- Let see example of AES and consider some of its implications.
- Although you are not expected to duplicate the example by hand, you will find it informative to study the hex patterns that occur from one step to the next.

Plaintext:	0123456789abcdeffedcba9876543210
Key:	0f1571c947d9e8590cb7add6af7f6798
Ciphertext:	Ff0b844a0853bf7c6934ab4364148fb9

- **Result:** Above table shows plain text, key and cipher text when we apply all the steps of AES we will get cipher text as shown.
- **The Avalanche Effect:** A desirable property of any encryption algorithm is that a small change in either the plaintext or the key should produce a significant change in cipher text.
- In particular, a change in one bit of plaintext or one bit of the key should produce a change in many bits of the ciphertext. This is referred to as the avalanche effect.
- In AES 1 bit change in input will affect nearly all bit of output after all rounds.

AES Implementation

Equivalent Inverse Cipher

- While implementing AES if we interchange the order of operation than it will affect the result or not is discussed here.
- If we interchange inverse shift row and inverse substitute byte operation than it will not affect and we get the same output.
- So we can obtain two equivalent decryption algorithms for one encryption algorithm.
- As inverse shift row will change position of byte and it will not affect byte value. While inverse substitute byte will change byte value by table lookup and it not concern with byte position. So we can interchange those two operations.
- If we interchange inverse mix column and add round key operation than it will affect and we do not get the same output.
- Both the operation will affect the value and so it cannot be interchange.

Implementation Aspects

- As in AES out of four three operation are byte level operation and it can be efficiently implemented on 8-bit processors.

- Only mix column operation is requiring matrix multiplication which requires some storage space and also time consuming on 8-bit processor.
- To overcome it we can use table lookup to reduce time requirement.
- Also we can implement it on 32-bit processors.
- In 32-bit processor we can use word by word operation and it much faster.

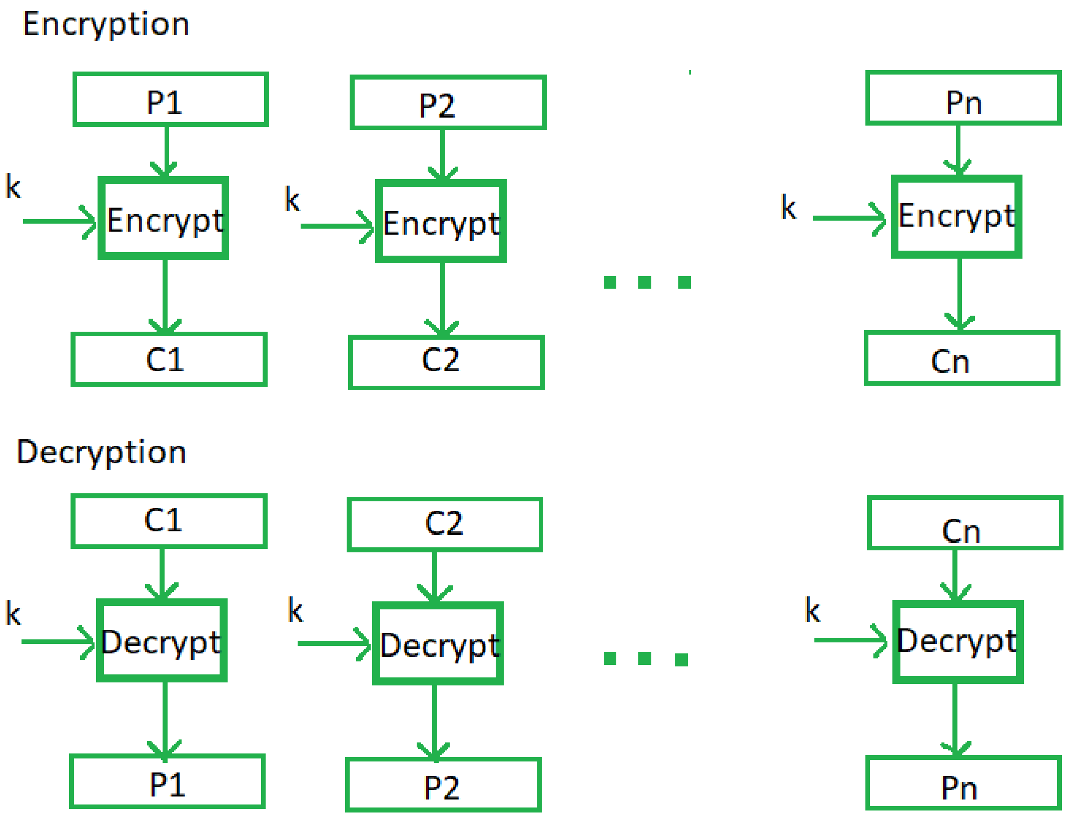
Block Cipher modes of Operation

Encryption algorithms are divided into two categories based on input type, as block cipher and stream cipher. **Block cipher** is an encryption algorithm which takes fixed size of input say *b* bits and produces a ciphertext of *b* bits again. If input is larger than *b* bits it can be divided further. For different applications and uses, there are several modes of operations for a block cipher.

Electronic Code Book (ECB) –

Electronic code book is the easiest block cipher mode of functioning. It is easier because of direct encryption of each block of input plaintext and output is in form of blocks of encrypted ciphertext. Generally, if a message is larger than *b* bits in size, it can be broken down into bunch of blocks and the procedure is repeated.

Procedure of ECB is illustrated below:



Advantages of using ECB –

- Parallel encryption of blocks of bits is possible, thus it is a faster way of encryption.
- Simple way of block cipher.

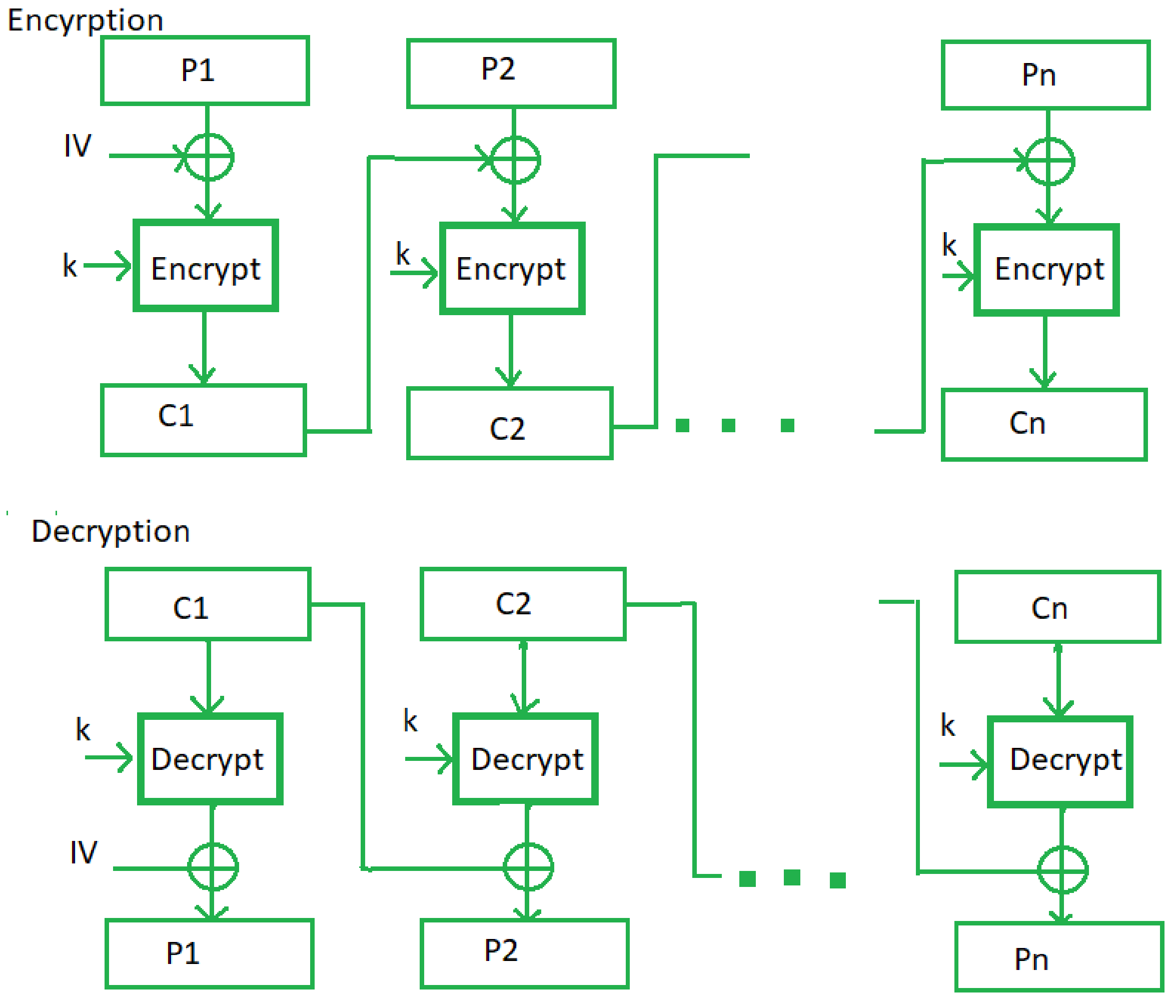
Disadvantages of using ECB –

- Prone to cryptanalysis since there is a direct relationship between plaintext and ciphertext.

Cipher Block Chaining –

Cipher block chaining or CBC is an advancement made on ECB since ECB compromises some security requirements. In CBC, previous cipher block is given as input to next encryption algorithm after XOR with original plaintext block. In a nutshell here, a cipher block is produced by encrypting a XOR output of previous cipher block and present plaintext block.

The process is illustrated here:



Advantages of CBC –

- CBC works well for input greater than *b* bits.
- CBC is a good authentication mechanism.

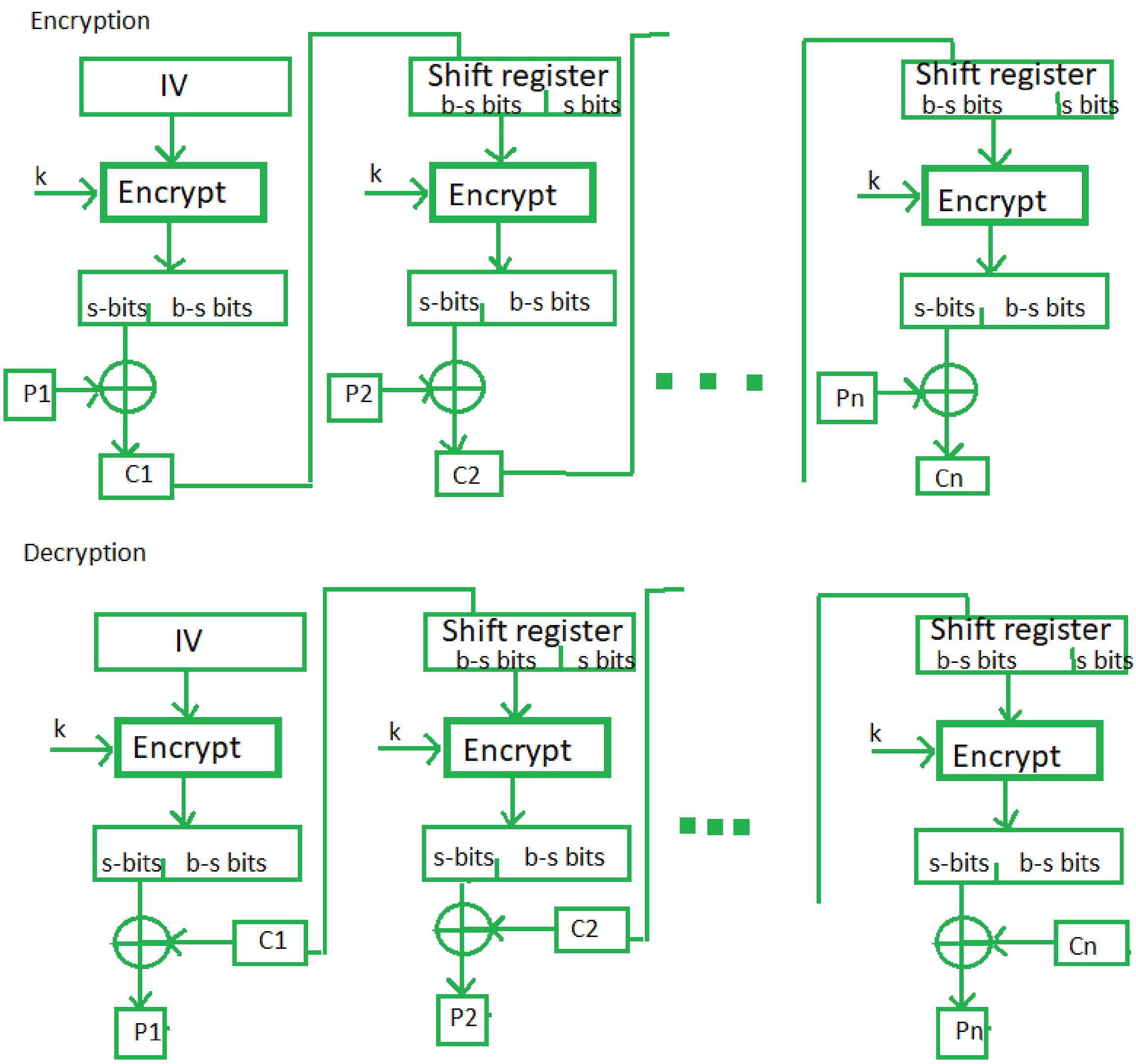
- Better resistive nature towards cryptanalsis than ECB.

Disadvantages of CBC –

- Parallel encryption is not possible since every encryption requires previous cipher.

Cipher Feedback Mode (CFB) –

In this mode the cipher is given as feedback to the next block of encryption with some new specifications: first an initial vector IV is used for first encryption and output bits are divided as set of *s*and*b-s* bits the left hand side *s*bits are selected and are applied an XOR operation with plaintext bits. The result given as input to a shift register and the process continues. The encryption and decryption process for the same is shown below, both of them use encryption algorithm.

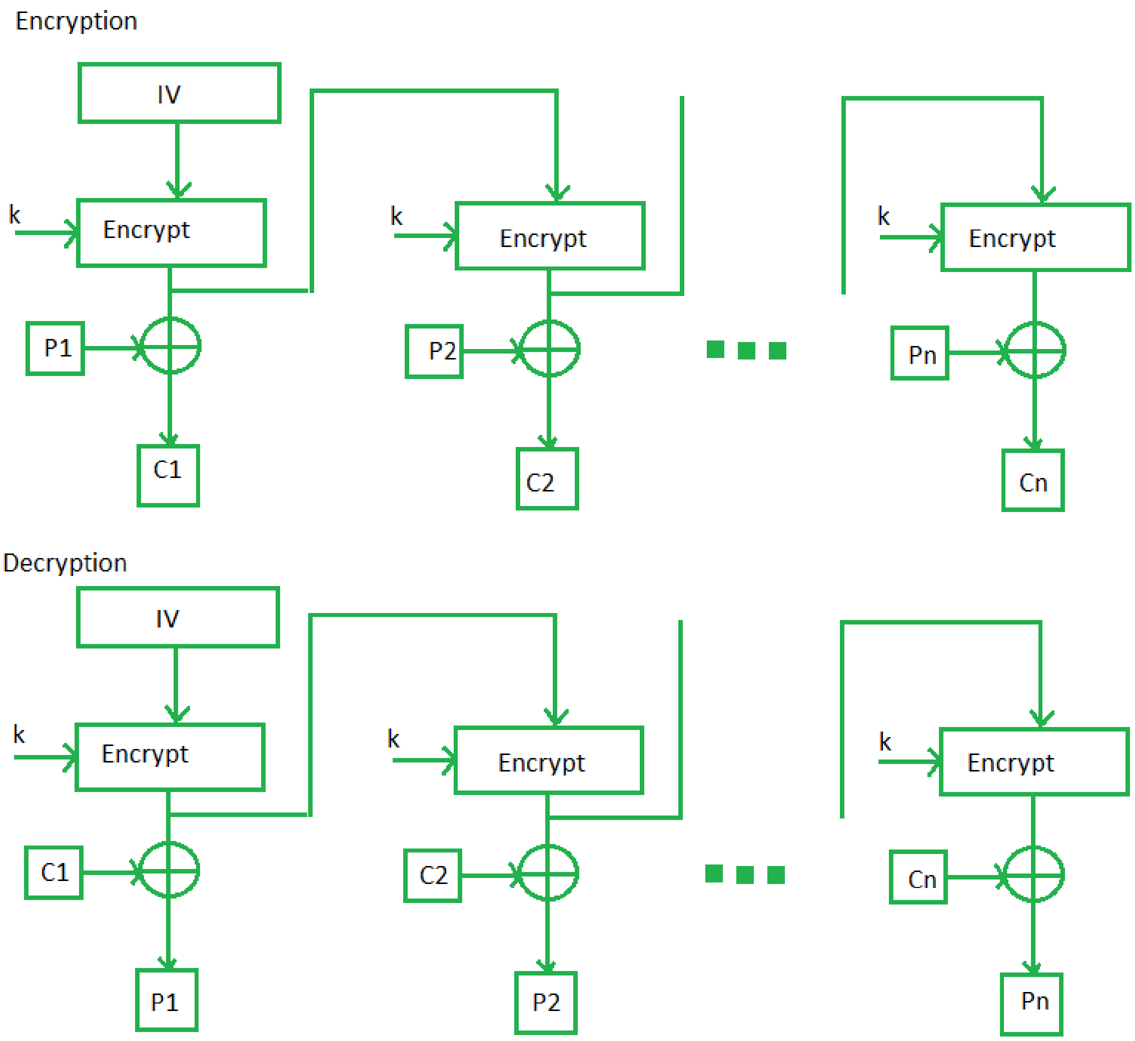


Advantages of CFB –

- Since, there is some data loss due to use of shift register, thus it is difficult for applying cryptanalysis.

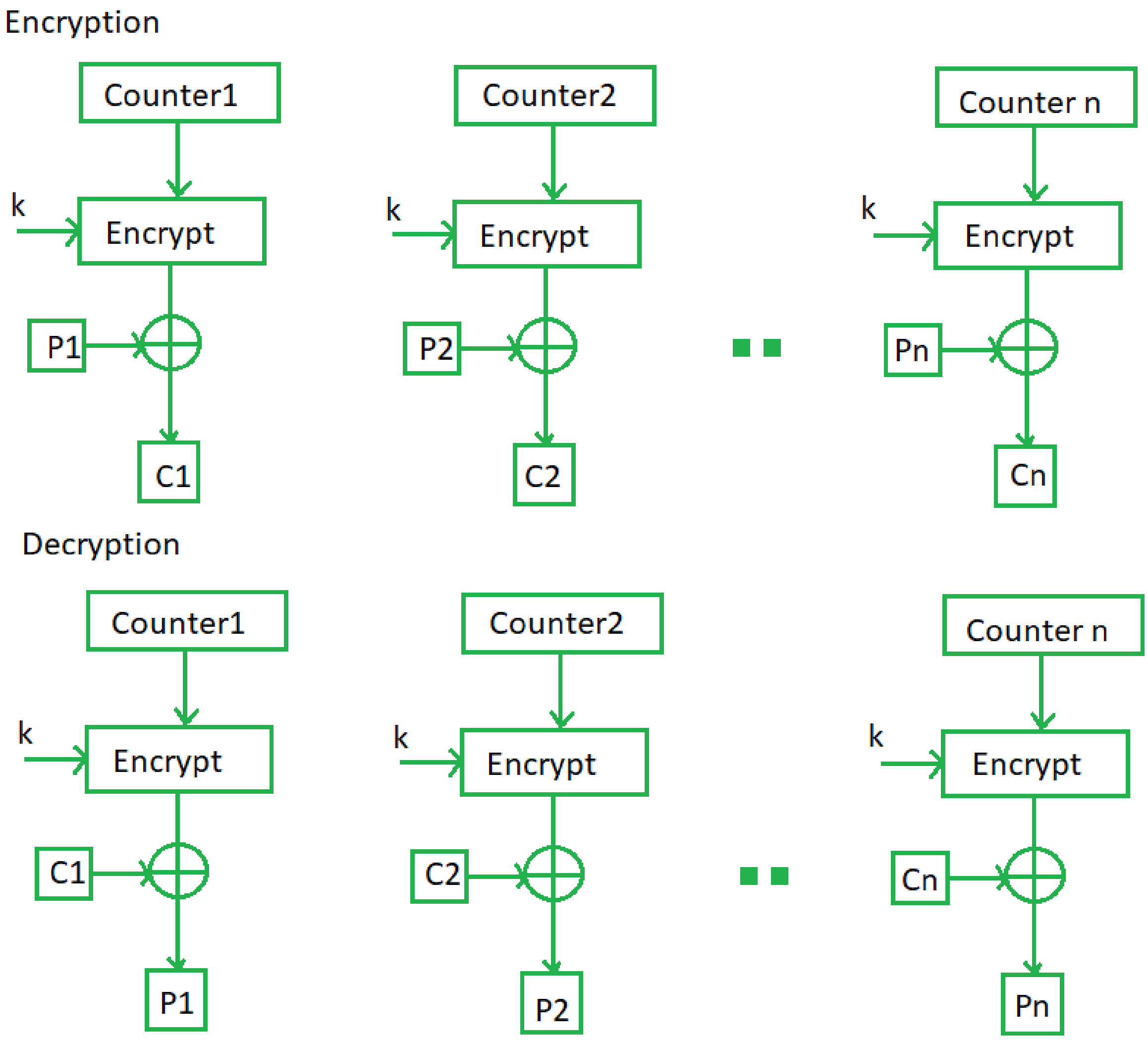
Output Feedback Mode –

The output feedback mode follows nearly same process as the Cipher Feedback mode except that it sends the encrypted output as feedback instead of the actual cipher which is XOR output. In this output feedback mode, all bits of the block are send instead of sending selected *s* bits. The Output Feedback mode of block cipher holds great resistance towards bit transmission errors. It also decreases dependency or relationship of cipher on plaintext.



Counter Mode –

The Counter Mode or CTR is a simple counter based block cipher implementation. Every time a counter initiated value is encrypted and given as input to XOR with plaintext which results in ciphertext block. The CTR mode is independent of feedback use and thus can be implemented in parallel.
Its simple implementation is shown below:



MODULE 2

INTERNATIONAL DATA ENCRYPTION ALGORITHM (IDEA)

IDEA (International Data Encryption Algorithm) was originally called IPES (Improved Proposed Encryption Standard). It was developed by Xuejia Lai and James L. Massey of ETH Zuria.

IDEA was designed to be efficient to compute in software. It encrypts a 64-bit block of plaintext into a 64-bit block of ciphertext using a 128-bit key. It was published in 1991, so cryptanalysts have had time to find weaknesses. So far none has been found, at least by the good guys (the ones who would publish their results).

IDEA is similar to DES in some ways. Both of them operate in rounds, and both have a complicated mangler function that does not have to be reversible in order for decryption to work. Instead, the mangler function is run in the same direction for encryption as decryption, in both IDEA and DES. In fact, both DES and IDEA have the property that encryption and decryption are identical except for key expansion. With DES, the same keys are used in the reverse order (with IDEA, the encryption and decryption keys are related in a more complex manner.

PRIMITIVE OPERATIONS

Each primitive operation in IDEA maps two 16-bit quantities into a 16-bit quantity. (In contrast, each DES S-box maps a 6-bit quantity into a 4-bit quantity.) IDEA uses three operations, all easy to compute in software, to create a mapping. Furthermore, the operations are all reversible, which is important in order to run IDEA backwards (i.e., to decrypt).

The operations are bitwise exclusive or ($\oplus$), a slightly modified add (+), and a slightly modified multiply ($\otimes$). The reason add and multiply can't be used in the ordinary way is that the result has to be 16 bits, which won't always be the case when adding or multiplying two 16-bit quantities. Addition in IDEA is done by throwing away carries, which is equivalent to saying addition is mod 2^{16}. Multiplication in IDEA is done by first calculating the 32-bit result, and then taking the remainder when divided by $2^{16}+1$ (which can be done in a clever efficient manner). Multiplication mod $2^{16}+1$ is reversible, in the sense that every number x between 1 and 2^{16} has an inverse y (i.e., a number in the range 1 to 2^{16} such that multiplication by y will “undo” multiplication by x), because $2^{16}+1$ happens to be prime. There is one subtlety, though. The number 0, which can be expressed in 16 bits, would not have an inverse. And the number 2^{16}, which is in the proper range for mod $2^{16}+1$ arithmetic, cannot be expressed in 16 bits. So both problems are solved by treating 0 as an encoding for 2^{16}.

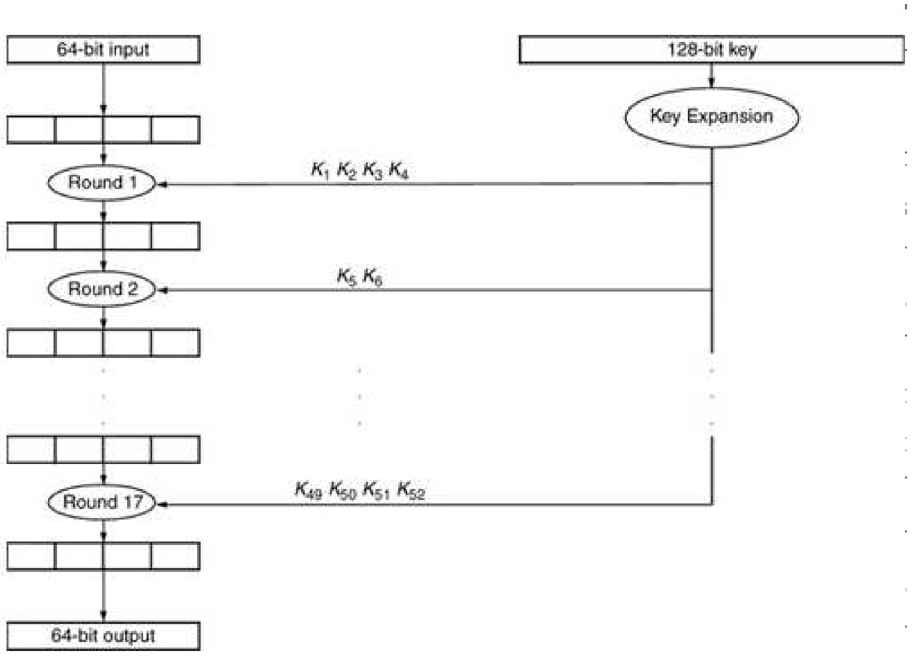


Figure 3-18. Basic Structure of IDEA

How are these operations reversible? Of course the operations are not reversible if all that is known is the 16-bit output. For instance, if we have inputs A and B, and perform $\oplus$ to obtain C, we can’t find A and B from C alone. However, when running IDEA backwards we will have C and B, and will use that to obtain A. $\oplus$ is easy. If you know B and C, then you can simply do $B \oplus C$ to get A. B is its own inverse with $\oplus$. + is easy, too. You compute $-B$ (mod 2^{16}). If you know C and $-B$, then you can find A by doing $C + -B$. With $\otimes$ you find B^{-1} (mod $2^{16}+1$) using Euclid’s algorithm and you perform $C \otimes B^{-1}$ to get A.

The only part of IDEA that isn’t necessarily reversible is the mangler function, and it is truly marvelous to note how IDEA’s design manages not to require a reversible mangler function

KEY EXPANSIONS

The 128-bit key is expanded into 52 16-bit keys, K_1, K_2,...K_{52}. The key expansion is done differently for encryption than for decryption. Once the 52 keys are generated, the encryption and decryption operations are the same.

The 52 encryption keys are generated by writing out the 128-bit key and, starting from the left, chopping off 16 bits at a time. This generates eight 16-bit keys (see Figure 3-19).

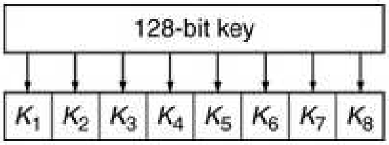


Figure 3-19. Generation of keys 1 through 8

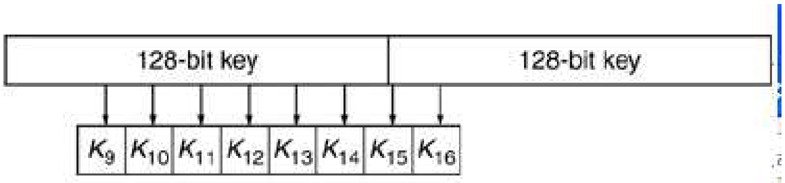

128-bit key
128-bit key
K_9 K_{10} K_{11} K_{12} K_{13} K_{14} K_{15} K_{16}

Figure 3-20. Generation of keys 9 through 16

The next eight keys are generated by starting at bit 25, and wrapping around to the beginning when the end is reached (see Figure 3-20).

The next eight keys are generated by offsetting 25 more bits, and so forth, until 52 keys are generated. The last offset starts at bit 23 and only needs 4 keys, so bits 1 thru 22 and bits 87 thru 128 get used in keys once less than bits 23 thru 86.

ONE ROUND

Like DES, IDEA is performed in rounds. It has 17 rounds, where the odd-numbered rounds are different from the even-numbered rounds. (Note that in other descriptions of IDEA, it is described as having 8 rounds, where those rounds do the work of two of our rounds. Our explanation is functionally equivalent. It's just that we think it's clearer to explain it as having 17 rounds.)

Each round takes the input, a 64-bit quantity, and treats it as four 16-bit quantities, which we'll call X_a, X_b, X_c, and X_d. Mathematical functions are performed on X_a, X_b, X_c, X_d to yield new versions of X_a, X_b, X_c, X_d.

The odd rounds use four of the K_i, which we'll call K_a, K_b, K_c, and K_d. The even rounds use two K_i, which we'll call K_e and K_f. So round one uses K_1, K_2, K_3, K_4 (i.e., in round 1, $K_a = K_1$, $K_b = K_2$, $K_c = K_3$, $K_d = K_4$). Round 2 uses K_5 and K_6 (i.e., in round 2, $K_e = K_5$ and $K_f = K_6$). Round 3 uses K_7, K_8, K_9, K_{10} ($K_a = K_7$ etc.). Round 4 uses K_{11} and K_{12}, and so forth.

An odd round, therefore has as input X_a, X_b, X_c, X_d and keys K_a, K_b, K_c, K_d. An even round has as input X_a, X_b, X_c, X_d and keys K_e and K_f.

ODD ROUND

The odd round is simple. X_a is replaced by $X_a \otimes K_a$. X_d is replaced by $X_d \otimes K_d$. X_c is replaced by $X_b + K_b$. X_b is replaced by $X_c + K_c$.

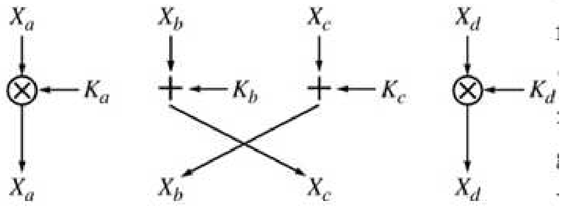


Figure 3-21. IDEA Odd Round

Note that this is easily reversible. To get from the new X_a to the old X_a, we perform $\otimes$ with the multiplicative inverse of K_a, mod 2^{16}+1. Likewise with X_d. To get the old X_b, given the new X_c, we add the additive inverse of K_b, i.e., we subtract K_b.

So when decrypting, the odd rounds run as before, but with the mathematical inverses of the keys. This will undo the work that was done during that round in encryption.

EVEN ROUND

The even round is a little more complicated. Again, we have X_a, X_b, X_c, and X_d. We have two keys, K_e and K_f. We're going to first compute two values, which we'll call Y_{in} and Z_{in}. We'll do a function, which we'll call the *mangler function*, which takes as input Y_{in}, Z_{in}, K_e, and K_f and produces what we'll call Y_{out} and Z_{out}. We'll use Y_{out} and Z_{out} to modify X_a, X_b, X_c, and X_d.

$\text{Yin} = \text{Xa} \oplus \text{Xb}$ $\text{Zin} = \text{Xc} \oplus \text{Xd}$

$\text{Yout} = ((\text{Ke} \otimes \text{Yin}) + \text{Zin}) \otimes \text{Kf}$ $\text{Zout} = (\text{Ke} \otimes \text{Yin}) + \text{Yout}$

Now we compute the new X_a, X_b, X_c, and X_d.

new $\text{Xa} = \text{Xa} \oplus \text{Yout}$ new $\text{Xb} = \text{Xb} \oplus \text{Yout}$

new $\text{Xc} = \text{Xc} \oplus \text{Zout}$ new $\text{Xd} = \text{Xd} \oplus \text{Zout}$

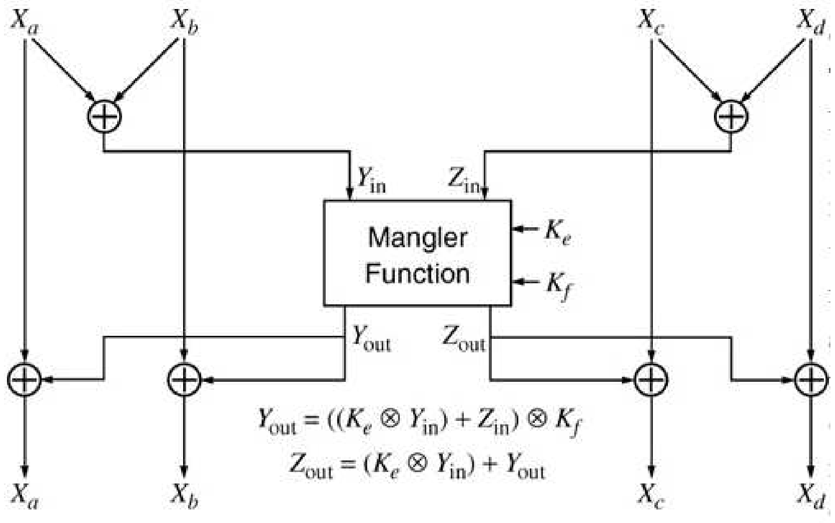

X_a
X_b
X_c
X_d
Y_{in}
Z_{in}
Mangler Function
K_e
K_f
Y_{out}
Z_{out}
$Y_{\text{out}} = ((K_e \otimes Y_{\text{in}}) + Z_{\text{in}}) \otimes K_f$
$Z_{\text{out}} = (K_e \otimes Y_{\text{in}}) + Y_{\text{out}}$
X_a
X_b
X_c
X_d

Figure 3-22. IDEA Even Round

How is the work of an even round reversed? This is truly spectacular (we don't get out much). The even round is its own inverse! When performing decryption, the same keys are used as when performing encryption (not the mathematical inverses of the keys, as in the odd rounds).

The even round takes as input the four quantities X_a, X_b, X_c, and X_d, together with keys K_e and K_f, and produces new X_a, new X_b, new X_c, and new X_d. If new X_a, new X_b, new X_c, and new X_d (with the same K_e and K_f) are fed into the even round, the output is the old X_a, X_b, X_c, and X_d. Why is this true?

Note that new $X_a = X_a \oplus \mathrm{Y}_{\mathrm{out}}$ and new $X_b = X_b \oplus \mathrm{Y}_{\mathrm{out}}$. In the beginning of the round, X_a and X_b are $\oplus$'d together, and the result is Y_{in}, the input to the mangler function. What if we use new X_a and new X_b instead of X_a and X_b? (new X_a) $\oplus$ (new X_b) = $(X_a \oplus \mathrm{Y}_{\mathrm{out}}) \oplus (X_b \oplus Y_{\mathrm{out}}) = X_a \oplus X_b$. So Y_{in} will be the same, whether X_a and X_b are the inputs, or new X_a and new X_b are the inputs. The same is true for X_c and X_d. (Z_{in} is the same value whether the inputs X_c and X_d are used, or inputs new X_c and new X_d are used.) So we've shown that the input to the mangler function is the same whether the input is X_a, X_b, X_c, and X_d or whether the input is new X_a, new X_b, new X_c, and new X_d.

That means the output of the mangler function will be the same whether you're doing encryption (starting with X_a, etc.) or decryption (starting with new X_a, etc.). We called the outputs of the mangler function Y_{out} and Z_{out}. To get the first output of the round (new X_a, in the case of encryption), we take the first input (X_a) and $\oplus$ it with Y_{out}. We're going to show that with inputs of new X_a, new X_b, new X_c, and new X_d, the output of the round is X_a, X_b, X_c, and X_d, i.e., that running the round with the output results in getting the input back.

We'll use as inputs new X_a, new X_b, new X_c, and new X_d, and we know that Y_{out} and Z_{out} are the same as they would have been with inputs of X_a, X_b, X_c, and X_d. What happens in the round? The first output of the round is computed by taking the first input and $\oplus$ing it with Y_{out}. We also know (from the encryption round) that new $X_a = X_a \oplus Y_{\mathrm{out}}$.

$$\text{first output} = \text{first input} \oplus \text{Yout}$$

$$\text{first output} = (\text{new Xa}) \oplus \text{Yout}$$

$$\text{first output} = (\text{Xa} \oplus \text{Yout}) \oplus \text{Yout} = \text{Xa}$$

With an input of new X_a, we get an output of X_a.

INVERSE KEYS FOR DECRYPTION

IDEA is cleverly designed so that the same code (or hardware) can perform either encryption or decryption given different expanded keys. We want to compute inverse keys such that the encryption procedure, unmodified, will work as a decryption procedure. The basic idea is to take the inverses of the encryption keys and use them in the opposite order (use the inverse of the last-used encryption key as the first key used when doing decryption).

Remember that for encryption, we generated 52 keys, K_1 through K_{52}. We use four of them in each of the odd rounds, and two of them in each of the even rounds. And since we are working backwards, the first decryption keys should be inverses of the last-used encryption keys. Given that the final keys used are K_{49}, K_{50}, K_{51}, and K_{52}, in an odd round, the first four decryption keys will be inverses of the keys K_{49}–K_{52}. K_{49} is used in $\otimes$, so the decryption key K_1 will be the multiplicative inverse of K_{49} mod $2^{16}+1$. And the decryption key K_4 is the multiplicative inverse of K_{52}. Decryption keys K_2 and K_3 are the additive inverses of K_{50} and K_{51} (meaning negative K_{50} and K_{51}).

In the even rounds, as we explained, the keys do not have to be inverted. The same keys are used for encryption as decryption.

ADVANCED ENCRYPTION STANDARD (AES)

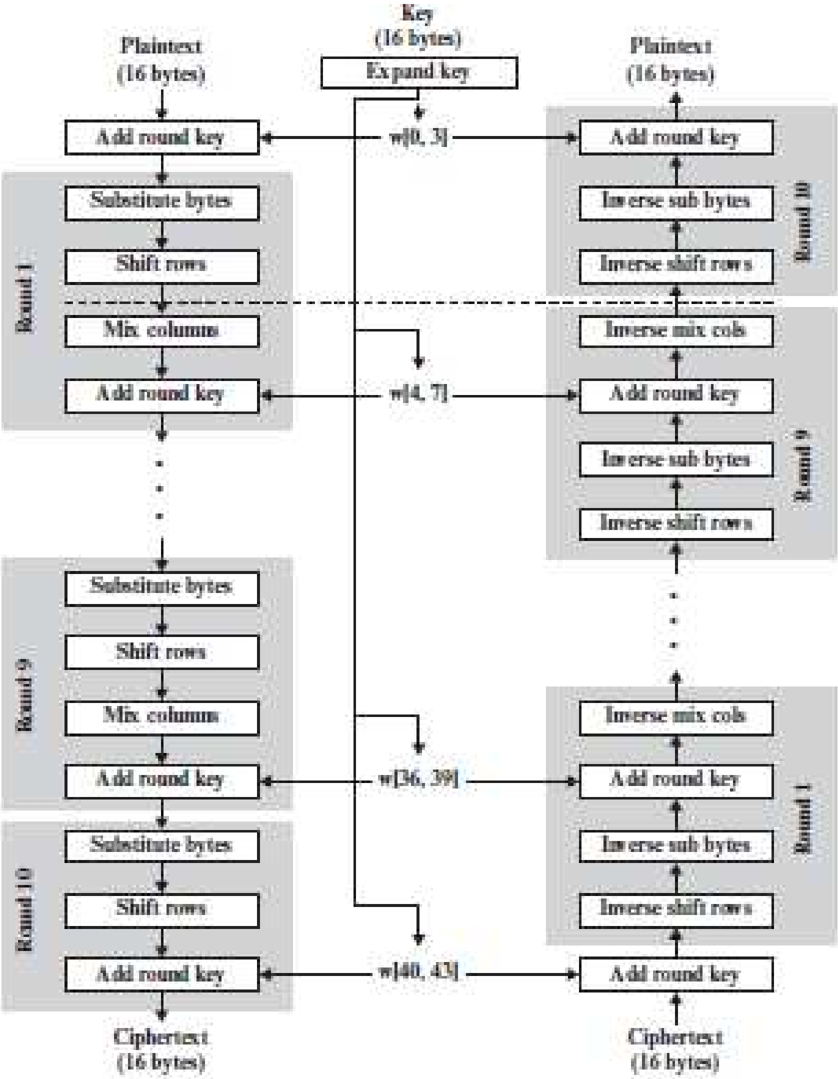
The world needed a new secret key standard. DES's key was too small. Triple DES was too slow. IDEA was encumbered (i.e., had patent protection), suspect, and slow. The National Institute of Standards and Technology (NIST) decided it wanted to facilitate creation of a new standard, but it had at least as difficult a political problem as a technical problem

After lots of investigation and discussion in the cryptographic community, NIST chose an algorithm called *Rijndael*, named after the two Belgian cryptographers who developed and submitted it—Dr. Joan Daemen of Proton World International and Dr. Vincent Rijmen, a postdoctoral researcher in the Electrical Engineering Department (ESAT) of Katholieke Universiteit Leuven [DAE99]. As of 26 November 2001, AES, a standardization of Rijndael, is a Federal Information Processing Standard [FIPS01].

Rijndael provides for a variety of block and key sizes. These two parameters can be chosen independently from 128, 160, 192, 224, and 256 bits. (in particular, key size and block size can be different). AES mandates a block size of 128 bits, and a choice of key size from 128, 192, and 256 bits [*], with the resulting versions imaginatively called AES-128, AES-192, and AES-256, respectively. We'll describe Rijndael, mentioning the AES parameters explicitly from time to time.

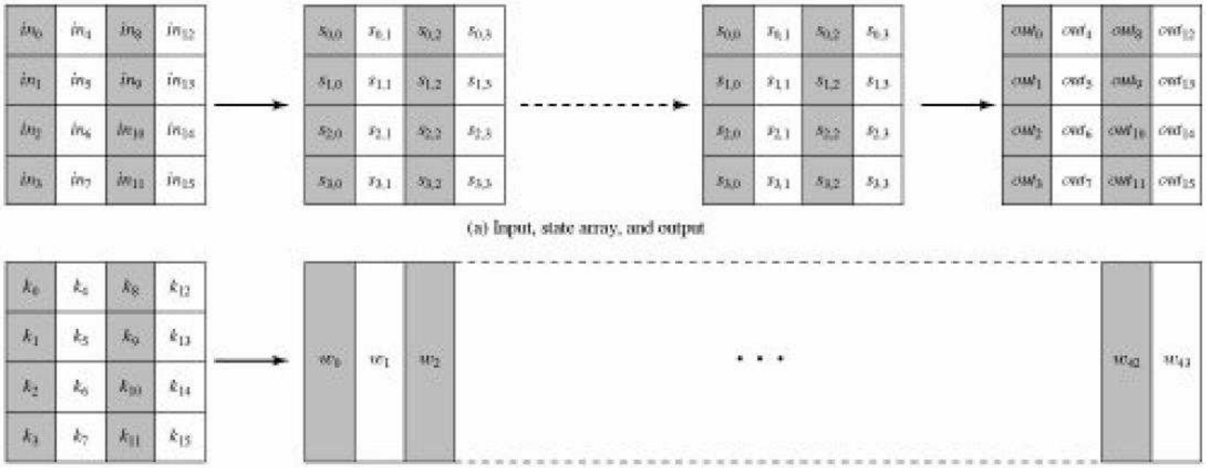
BASIC STRUCTURE

Figure 5.1 shows the overall structure of AES. The input to the encryption and decryption algorithms is a single 128-bit block. This block is copied into the **State** array, which is modified at each stage of encryption or decryption. After the final stage, **State** is copied to an output matrix. Similarly, the 128-bit key is depicted as a square matrix of bytes. This key is then expanded into an array of key schedule words; each word is four bytes and the total key schedule is 44 words for the 128-bit key.Note that the ordering of bytes within a matrix is by column. So, for example, the first four bytes of a 128-bit plaintext input to the encryption cipher occupy the first column of the **in** matrix, the second four bytes occupy the second column, and so on. Similarly, the first four bytes of the expanded key, which form a word, occupy the first column of the **w** matrix.



(a) Encryption (b) Decryption

Figure 6.3 AES Encryption and Decryption


in_0 in_4 in_8 in_{12}
in_1 in_5 in_9 in_{13}
in_2 in_6 in_{10} in_{14}
in_3 in_7 in_{11} in_{15}
$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,0}$ $s_{1,1}$ $s_{1,2}$ $s_{1,3}$
$s_{2,0}$ $s_{2,1}$ $s_{2,2}$ $s_{2,3}$
$s_{3,0}$ $s_{3,1}$ $s_{3,2}$ $s_{3,3}$
out_0 out_4 out_8 out_{12}
out_1 out_5 out_9 out_{13}
out_2 out_6 out_{10} out_{14}
out_3 out_7 out_{11} out_{15}
k_0 k_4 k_8 k_{12}
k_1 k_5 k_9 k_{13}
k_2 k_6 k_{10} k_{14}
k_3 k_7 k_{11} k_{15}
w_0 w_1 w_2 . . . w_{42} w_{43}

(a) Input, state array, and output

(b) Key and expanded key

PRIMITIVE OPERATION

Four different stages are used, one of permutation and three of substitution:

■ **Substitute bytes:** Uses an S-box to perform a byte-by-byte substitution of the block.

■ **ShiftRows:** A simple permutation.

■ **MixColumns:** A substitution that makes use of arithmetic over GF(28).

■ **AddRoundKey:** A simple bitwise XOR of the current block with a portion of the expanded key

For both encryption and decryption, the cipher begins with an AddRoundKey stage, followed by nine rounds that each includes all four stages, followed by a tenth round of three stages. Only the AddRoundKey stage makes use of the key. For this reason, the cipher begins and ends with an AddRoundKey stage. Any other stage, applied at the beginning or end, is reversible without knowledge of the key and so would add no security.

The AddRoundKey stage is, in effect, a form of Vernam cipher and by itself would not be formidable. The other three stages together provide confusion, diffusion, and nonlinearity, but by themselves would provide no security because they do not use the key. We can view the cipher as alternating operations of XOR encryption (AddRoundKey) of a block, followed by scrambling of the block (the other three stages), followed by XOR encryption, and so on. This scheme is both efficient and highly secure.

Each stage is easily reversible. For the Substitute Byte, ShiftRows, and MixColumns stages, an inverse function is used in the decryption algorithm. For the AddRoundKey stage, the inverse is achieved by XORing the same round key to the block. As with most block ciphers, the decryption algorithm makes use of the expanded key in reverse order. However, the decryption algorithm is not identical to the encryption algorithm. This is a consequence of the particular structure of AES.

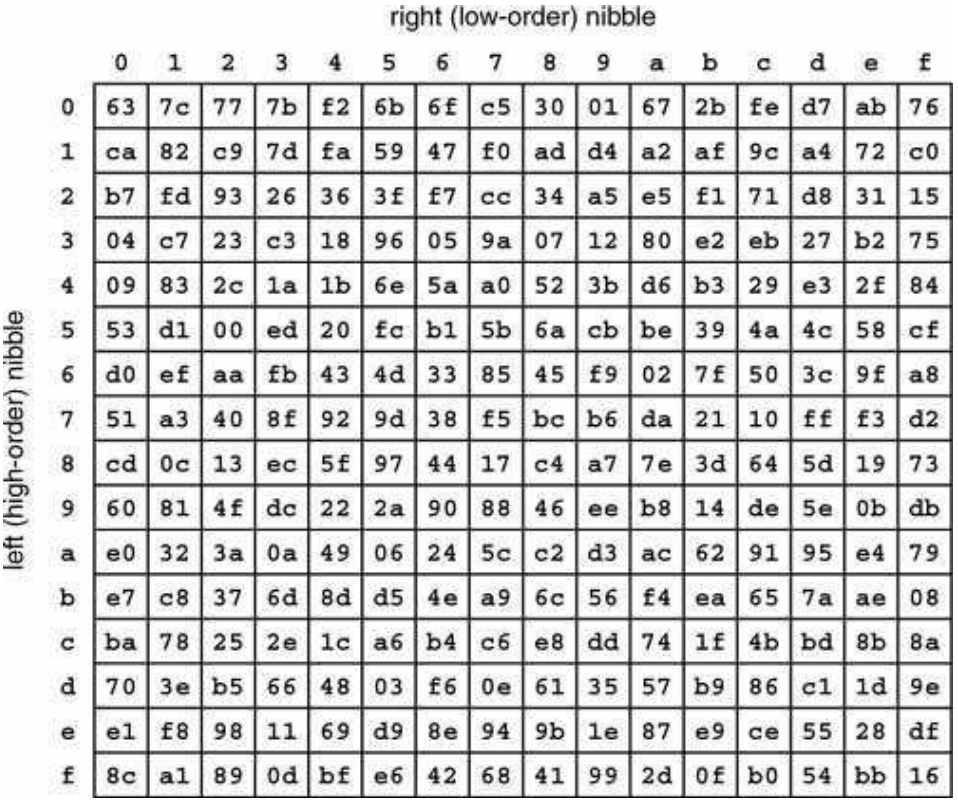
Once it is established that all four stages are reversible, it is easy to verify that decryption does recover the plaintext. Figure 5.1 lays out encryption and decryption going in opposite vertical directions. At each horizontal point (e.g., the dashed line in the figure), **State** is the same for both encryption and decryption.

The final round of both encryption and decryption consists of only three stages. Again, this is a consequence of the particular structure of AES and is required to make the cipher reversible.

Substitute Bytes Transformation

Forward and Inverse Transformations

The **forward substitute byte transformation**, called SubBytes, is a simple table lookup AES defines a 16 x 16 matrix of byte values, called an S-box (Table 5.4a), that contains a permutation of all possible 256 8-bit values. Each individual byte of **State** is mapped into a new byte in the following way: The leftmost 4 bits of the byte are used as a row value and the rightmost 4 bits are used as a column value. These row and column values serve as indexes into the S-box to select a unique 8-bit output value. For example, the hexadecimal value {95} references row 9, column 5 of the S-box, which contains the value {2A}. Accordingly, the value {95} is mapped into the value {2A}.

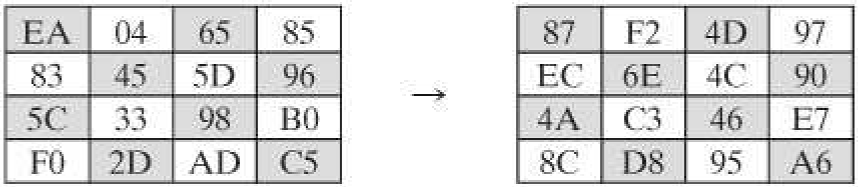


right (low-order) nibble

left (high-order) nibble

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Figure 3-24. Rijndael S-box



EA	04	65	85
83	45	5D	96
5C	33	98	B0
F0	2D	AD	C5

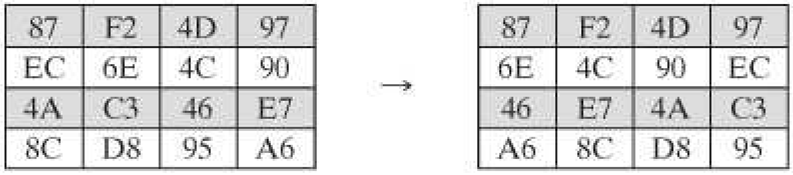
→

87	F2	4D	97
EC	6E	4C	90
4A	C3	46	E7
8C	D8	95	A6

ShiftRows Transformation

Forward and Inverse Transformations

The **forward shift row transformation**, called ShiftRows. The first row of **State** is not altered. For the second row, a 1-byte circular left shift is performed. For the third row, a 2-byte circular left shift is performed. For the fourth row, a 3-byte circular left shift is performed. The following is an example of ShiftRows:



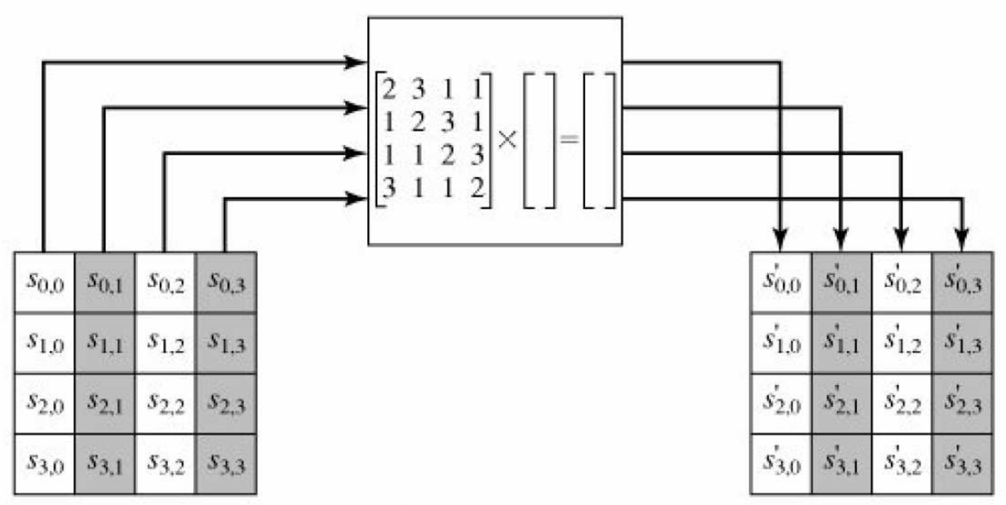
87	F2	4D	97
EC	6E	4C	90
4A	C3	46	E7
8C	D8	95	A6

→

87	F2	4D	97
6E	4C	90	EC
46	E7	4A	C3
A6	8C	D8	95

Mix Column Transformation

The **forward mix column transformation**, called MixColumns, operates on each column individually. Each byte of a column is mapped into a new value that is a function of all four bytes in that column. The transformation can be defined by the following matrix multiplication on **State**



$\begin{bmatrix}2&3&1&1\\1&2&3&1\\1&1&2&3\\3&1&1&2\end{bmatrix} \times \begin{bmatrix} \\ \end{bmatrix} = \begin{bmatrix} \\ \end{bmatrix}$
$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,0}$ $s_{1,1}$ $s_{1,2}$ $s_{1,3}$
$s_{2,0}$ $s_{2,1}$ $s_{2,2}$ $s_{2,3}$
$s_{3,0}$ $s_{3,1}$ $s_{3,2}$ $s_{3,3}$
$s'_{0,0}$ $s'_{0,1}$ $s'_{0,2}$ $s'_{0,3}$
$s'_{1,0}$ $s'_{1,1}$ $s'_{1,2}$ $s'_{1,3}$
$s'_{2,0}$ $s'_{2,1}$ $s'_{2,2}$ $s'_{2,3}$
$s'_{3,0}$ $s'_{3,1}$ $s'_{3,2}$ $s'_{3,3}$

(b) Mix column transformation

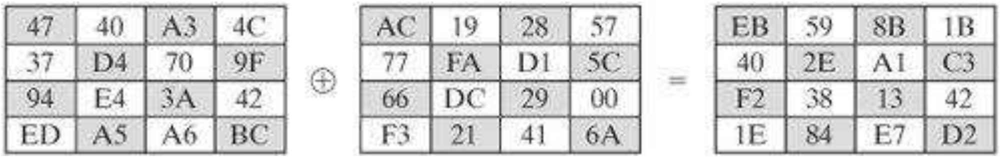
Each element in the product matrix is the sum of products of elements of one row and one column. In this case, the individual additions and multiplications are performed in GF(2^8). Use the symbol · to indicate multiplication over the finite field GF(2^8) and ⊕ to indicate bitwise XOR, which corresponds to addition in GF(2^8).

$$
\begin{aligned}
s'_{0,j} &= (2 \bullet s_{0,j}) \oplus (3 \bullet s_{1,j}) \oplus s_{2,j} \oplus s_{3,j} \\
s'_{1,j} &= s_{0,j} \oplus (2 \bullet s_{1,j}) \oplus (3 \bullet s_{2,j}) \oplus s_{3,j} \\
s'_{2,j} &= s_{0,j} \oplus s_{1,j} \oplus (2 \bullet s_{2,j}) \oplus (3 \bullet s_{3,j}) \\
s'_{3,j} &= (3 \bullet s_{0,j}) \oplus s_{1,j} \oplus s_{2,j} \oplus (2 \bullet s_{3,j})
\end{aligned}
$$

AddRoundKey Transformation

Forward and Inverse Transformations

In the **forward add round key transformation**, called AddRoundKey, the 128 bits of **State** are bitwise XORed with the 128 bits of the round key. As shown in Figure 5.4b, the operation is viewed as a columnwise operation between the 4 bytes of a **State** column and one word of the round key; it can also be viewed as a byte-level operation. The following is an example of AddRoundKey:



$$
\begin{bmatrix} 47 & 40 & A3 & 4C \\ 37 & D4 & 70 & 9F \\ 94 & E4 & 3A & 42 \\ ED & A5 & A6 & BC \end{bmatrix}
\oplus
\begin{bmatrix} AC & 19 & 28 & 57 \\ 77 & FA & D1 & 5C \\ 66 & DC & 29 & 00 \\ F3 & 21 & 41 & 6A \end{bmatrix}
=
\begin{bmatrix} EB & 59 & 8B & 1B \\ 40 & 2E & A1 & C3 \\ F2 & 38 & 13 & 42 \\ 1E & 84 & E7 & D2 \end{bmatrix}
$$

AES Key Expansion

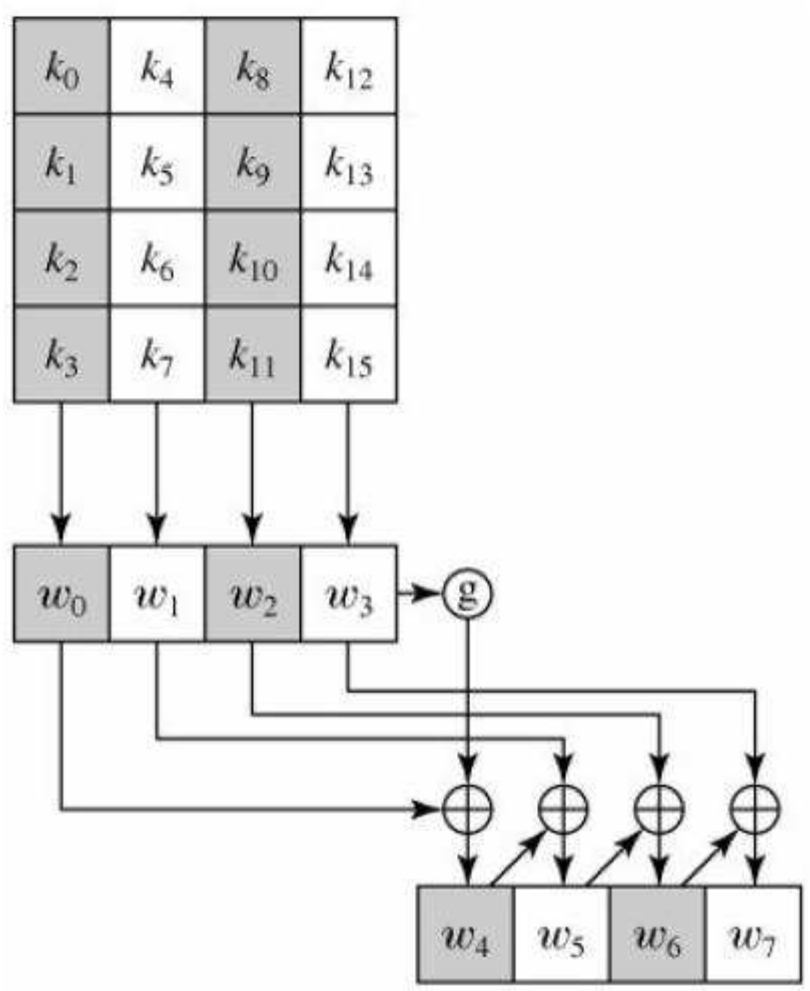
Key Expansion Algorithm

The AES key expansion algorithm takes as input a 4-word (16-byte) key and produces a linear array of 44 words (176 bytes). The key is copied into the first four words of the expanded key. The remainder of the expanded key is filled in four words at a time. Each added word **w**[i] depends on the immediately preceding word, **w**[i1], and the word four positions back,**w**[i 4]. In three out of four cases, a simple XOR is used. For a word whose position in the **w** array is a multiple of 4, a more complex function is used. Figure illustrates the generation of the first eight words of the expanded key, using the symbol g to represent that complex function. The function g consists of the following subfunctions:

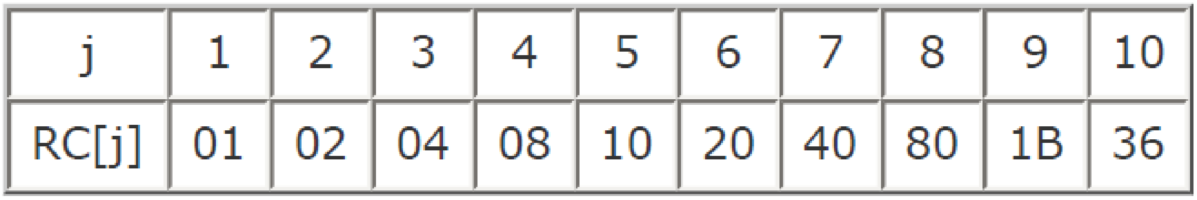
1.RotWord performs a one-byte circular left shift on a word. This means that an input word [b0,b1, b2, b3] is transformed into [b1, b2, b3, b0].

2.SubWord performs a byte substitution on each byte of its input word, using the S-box

3.The result of steps 1 and 2 is XORed with a round constant, Rcon[j].



The round constant is a word in which the three rightmost bytes are always 0. Thus the effect of an XOR of a word with Rcon is to only perform an XOR on the leftmost byte of the word. The round constant is different for each round and is defined as Rcon[j] = (RC[j], 0, 0, 0), with RC[1] = 1, RC[j] = 2 · RC[j -1] and with multiplication defined over the field GF(2^8). The values of RC[j] in hexadecimal are



j	1	2	3	4	5	6	7	8	9	10
RC[j]	01	02	04	08	10	20	40	80	1B	36

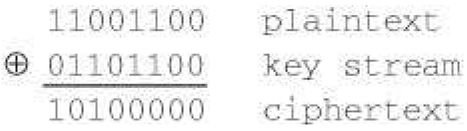
Stream Ciphers and RC4

In this section we look at perhaps the most popular symmetric stream cipher, RC4. We begin with an overview of stream cipher structure, and then examine RC4.

STREAM CIPHER STRUCTURE

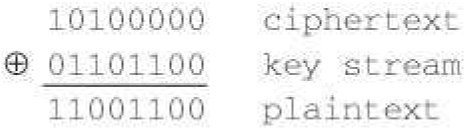
A typical stream cipher encrypts plaintext one byte at a time, although a stream cipher may be designed to operate on one bit at a time or on units larger than a byte at a time. Figure 6.8 is a representative diagram of stream cipher structure. In this structure a key is input to a pseudorandom bit generator that produces a stream of 8-bit numbers that are apparently random. We discuss pseudorandom number generators in Chapter 7. For now, we simply say that a pseudorandom stream is one that is unpredictable without knowledge of the input key. The output of the generator, called a **keystream**, is combined one byte at a time with the plaintext stream using the bitwise exclusive-OR (XOR) operation.

For example, if the next byte generated by the generator is 01101100 and the next plaintext byte is 11001100, then the resulting ciphertext byte is



```
  11001100   plaintext
⊕ 01101100   key stream
  10100000   ciphertext
```

Decryption requires the use of the same pseudorandom sequence:



```
  10100000   ciphertext
⊕ 01101100   key stream
  11001100   plaintext
```

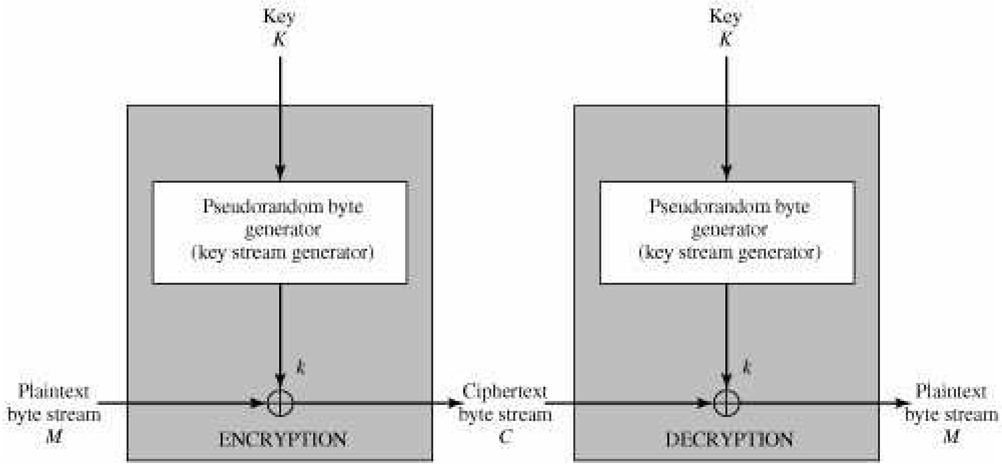


Figure 6.8. Stream Cipher Diagram

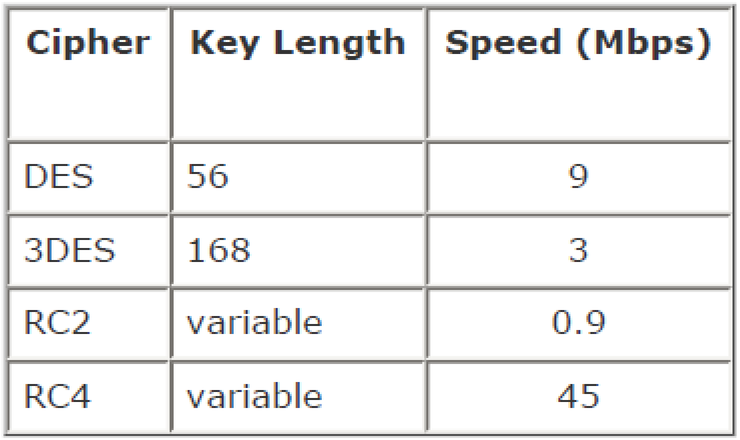
The stream cipher is similar to the one-time pad discussed in Chapter 2. The difference is that a onetime pad uses a genuine random number stream, whereas a stream cipher uses a pseudorandom number stream.

[KUMA97] lists the following important design considerations for a stream cipher:

1. The encryption sequence should have a large period. A pseudorandom number generator uses a function that produces a deterministic stream of bits that eventually repeats. The longer the period of repeat the more difficult it will be to do cryptanalysis. This is essentially the same consideration that was discussed with reference to the Vigenère cipher, namely that the longer the keyword the more difficult the cryptanalysis.
2. The keystream should approximate the properties of a true random number stream as close as possible. For example, there should be an approximately equal number of 1s and 0s. If the keystream is treated as a stream of bytes, then all of the 256 possible byte values should appear approximately equally often. The more random-appearing the keystream is, the more randomized the ciphertext is, making cryptanalysis more difficult.
3. Note from Figure 6.8 that the output of the pseudorandom number generator is conditioned on the value of the input key. To guard against brute-force attacks, the key needs to be sufficiently long. The same considerations as apply for block ciphers are valid here. Thus, with current technology, a key length of at least 128 bits is desirable.

With a properly designed pseudorandom number generator, a stream cipher can be as secure as block cipher of comparable key length. The primary advantage of a stream cipher is that stream ciphers are almost always faster and use far less code than do block ciphers. The example in this section, RC4, can be implemented in just a few lines of code. Table 6.2, using data from [RESC01], compares execution times of RC4 with three well-known symmetric block ciphers. The advantage of a block cipher is that you can reuse keys. However, if two plaintexts are encrypted with the same key using a stream cipher, then cryptanalysis is often quite simple [DAWS96]. If the two ciphertext streams are XORed together, the result is the XOR of the original plaintexts. If the plaintexts are text strings, credit card numbers, or other byte streams with known properties, then cryptanalysis may be successful.

Table 6.2. Speed Comparisons of Symmetric Ciphers on a Pentium II



Cipher	Key Length	Speed (Mbps)
DES	56	9
3DES	168	3
RC2	variable	0.9
RC4	variable	45

For applications that require encryption/decryption of a stream of data, such as over a data communications channel or a browser/Web link, a stream cipher might be the better alternative. For applications that deal with blocks of data, such as file transfer, e-mail, and database, block ciphers may be more appropriate. However, either type of cipher can be used in virtually any application.

RC4

A long random (or pseudo-random) string used to encrypt a message with a simple ⊕ operation is known as a *one-time pad*. A **stream cipher** generates a one-time *pad* and applies it to a stream of plaintext with ⊕.

RC4 is a stream cipher designed by Ron Rivest. RC4 was a trade secret, but was “outed” in 1994. As a result, it has been extensively analyzed and is considered secure as long as you discard the first few (say 256) octets of the generated pad.

The algorithm is an extremely simple (and fast) generator of pseudo-random streams of octets. The key can be from 1 to 256 octets. Even with a minimal key (a single null octet), the generated pseudo-random stream passes all the usual randomness tests (and so makes a fine pseudo-random number generator—I$_3$’ve used it for that purpose on numerous occasions). RC4 keeps 258 octets of state information, 256 octets of which are a permutation of 0, 1,...255 that is initially computed from the key and then altered as each pad octet is generated.

The box on page 93 gives a complete C implementation of RC4 one-time pad generation.

The RC4 Algorithm

RC4 is a stream cipher designed in 1987 by Ron Rivest for RSA Security. It is a variable key-size stream cipher with byte-oriented operations. The algorithm is based on the use of a random permutation.

Analysis shows that the period of the cipher is overwhelmingly likely to be greater than 10100 [ROBS95a]. Eight to sixteen machine operations are required per output byte, and the cipher can be expected to run very quickly in software. RC4 is used in the SSL/TLS (Secure Sockets Layer/Transport Layer Security) standards that have been defined for communication between Web browsers and servers. It is also used in the WEP (Wired Equivalent Privacy) protocol and the newer WiFi Protected Access (WPA) protocol that are part of the IEEE 802.11 wireless LAN standard. RC4 was kept as a trade secret by RSA Security. In September 1994, the RC4 algorithm was anonymously posted on the Internet on the Cypherpunks anonymous remailers list.

The RC4 algorithm is remarkably simply and quite easy to explain. A variable-length key of from 1 to 256 bytes (8 to 2048 bits) is used to initialize a 256-byte state vector S, with elements S[0], S[1],..., S[255]. At all times, S contains a permutation of all 8-bit numbers from 0 through 255. For encryption and decryption, a byte *k* (see Figure 6.8) is generated from S by selecting one of the 255 entries in a systematic fashion. As each value of *k* is generated, the entries in S are once again permuted.

Initialization of S

To begin, the entries of S are set equal to the values from 0 through 255 in ascending order; that is; S[0] = 0, S[1] = 1,..., S[255] = 255. A temporary vector, T, is also created. If the length of the key K is 256 bytes, then K is transferred to T. Otherwise, for a key of length *keylen* bytes, the first *keylen* elements of T are copied from K and then K is repeated as many times as necessary to fill out T. These preliminary operations can be summarized as follows:

```
/* Initialization */
for i = 0 to 255 do
    S[i] = i;
    T[i] = K[i mod keylen];
```

Next we use T to produce the initial permutation of S. This involves starting with S[0] and going through to S[255], and, for each S[i], swapping S[i] with another byte in S according to a scheme dictated by T[i]:

```
/* Initial Permutation of S */

j = 0;
for i = 0 to 255 do
    j = (j + S[i] + T[i]) mod 256;
    Swap (S[i], S[j]);
```

Because the only operation on S is a swap, the only effect is a permutation. S still contains all the numbers from 0 through 255.

Stream Generation

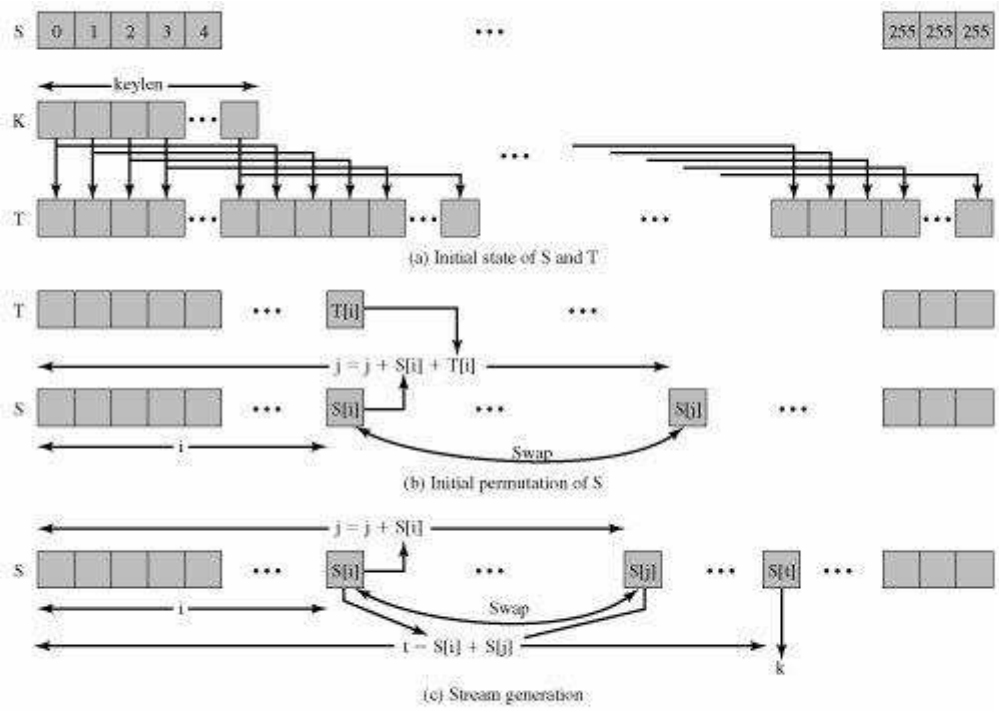
Once the S vector is initialized, the input key is no longer used. Stream generation involves cycling through all the elements of S[i], and, for each S[i], swapping S[i] with another byte in S according to a scheme dictated by the current configuration of S. After S[255] is reached, the process continues, starting over again at S[0]:

```
/* Stream Generation */

i, j = 0;
while (true)
i = (i + 1) mod 256;
j = (j + S[i]) mod 256;
Swap (S[i], S[j]);
t = (S[i] + S[j]) mod 256;
k = S[t];
```

To encrypt, XOR the value *k* with the next byte of plaintext. To decrypt, XOR the value *k* with the next byte of ciphertext.

Figure 6.9 illustrates the RC4 logic.



(a) Initial state of S and T

(b) Initial permutation of S

(c) Stream generation

Strength of RC4

A number of papers have been published analyzing methods of attacking RC4 [e.g., [KNUD98], [MIST98], [FLUH00], [MANT01]). None of these approaches is practical against RC4 with a reasonable key length, such as 128 bits. A more serious problem is reported in [FLUH01]. The authors demonstrate that the WEP protocol, intended to provide confidentiality on 802.11 wireless LAN networks, is vulnerable to a particular attack approach. In essence, the problem is not with RC4 itself but the way in which keys are generated for use as input to RC4. This particular problem does not appear to be relevant to other applications using RC4 and can be remedied in WEP by changing the way in which keys are generated. This problem points out the difficulty in designing a secure system that involves both cryptographic functions and protocols that make use of them.

Why Public-Key Cryptography?

- developed to address two key issues:
    - key distribution – how to have secure communications in general without having to trust a KDC with your key
    - digital signatures – how to verify a message comes intact from the claimed sender

Public-Key Characteristics

- **computationally infeasible to find decryption key** knowing only algorithm & encryption key
- **either of the two related keys can be used for encryption**, with the **other used for decryption** (in some schemes)

Public-Key Applications

- can classify uses into 3 categories:
    1. encryption/decryption (provide secrecy)
    2. digital signatures (provide authentication)
    3. key exchange (of session/secret keys)
- some algorithms are suitable for all uses, others are specific to one.

Public-Key Cryptography provides

- **Confidentiality/ Secrecy**
- **Authentication**

Public-Key Cryptosystems (provides secrecy) as follows

- Plain text X=[X1, X2,…Xm]
- Cipher text Y= [Y1, Y2…Yn]
- Let Message is from A to B
- Key pair of A (PRa, PUa)
- Key Pair of B (PRb, PUb)
- Then Y= E(Pub,X)
    - X=D(PRb, Y) provides secrecy

Public-Key Cryptosystems provides authentication and secrecy as follows

- Plain text X=[X1, X2,…Xm]
- Cipher text Y= [Y1, Y2…Yn]
- Let Message is from A to B
- Key pair of A (PRa, PUa)
- Key Pair of B (PRb, PUb)
- Then Z= E(PUb,E(PRa,X))
- X=D(PUa, D(PRb,Z))

CS409 CRYPTOGRAPHY AND NETWORK SECURITY

Module 3 (7 hrs)
Public key Cryptography: - Principles of Public key Cryptography Systems, Number theory-Fundamental Theorem of arithmetic, Fermat's Theorem, Euler's Theorem, Euler's Totient Function, Extended Euclid's Algorithm, Modular arithmetic. RSA algorithmKey Management - Diffie-Hellman Key Exchange, Elliptic curve cryptography

Topic 1 : Public key Cryptography

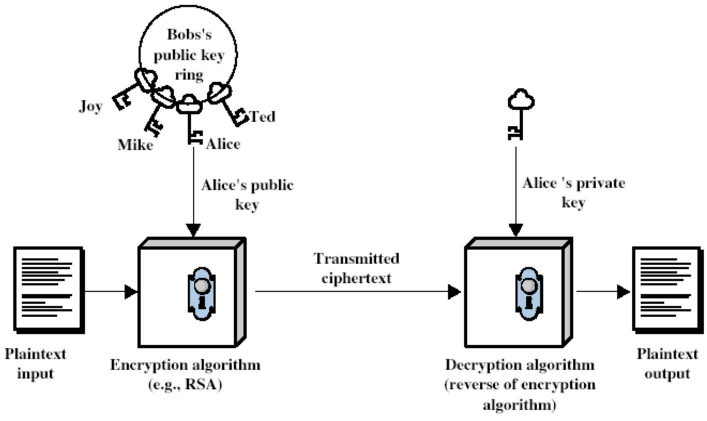
1. To explore the need of public key cryptosystem.
2. To introduce public key cryptosystem and its applications.
3. To Identify the Principles of Public Key Cryptosystems

3.1 Private-Key Cryptography

- traditional **private/secret/single key** cryptography uses **one** key
- shared by both sender and receiver
- also is **symmetric**, parties are equal

Public-Key Cryptography

- **public-key/two-key/asymmetric** cryptography involves the use of **two** keys:
    - a **public-key**, which may be known by anybody, and can be used to **encrypt messages**, and **verify signatures**
    - a **private-key**, known only to the recipient, used to **decrypt messages**, and **sign** (create) **signatures**
- is **asymmetric** because
    - those who encrypt messages or verify signatures **cannot** decrypt messages or create signatures.



- provides authentication and secrecy

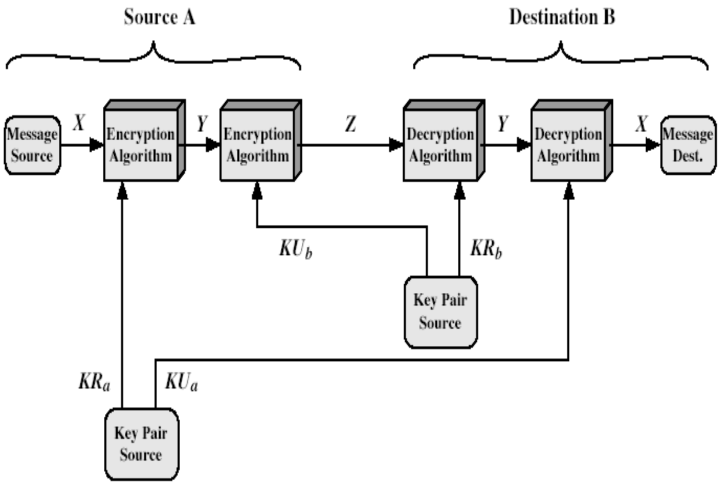
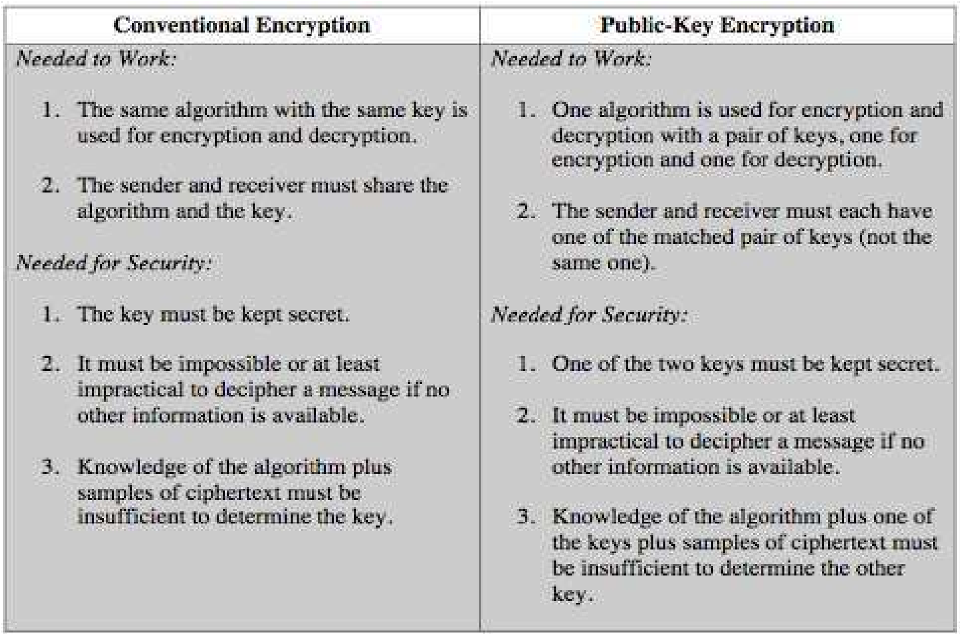


Figure 9.4 Public-Key Cryptosystem: Secrecy and Authentication



Conventional Encryption	Public-Key Encryption
Needed to Work: 1. The same algorithm with the same key is used for encryption and decryption. 2. The sender and receiver must share the algorithm and the key. *Needed for Security:* 1. The key must be kept secret. 2. It must be impossible or at least impractical to decipher a message if no other information is available. 3. Knowledge of the algorithm plus samples of ciphertext must be insufficient to determine the key.	*Needed to Work:* 1. One algorithm is used for encryption and decryption with a pair of keys, one for encryption and one for decryption. 2. The sender and receiver must each have one of the matched pair of keys (not the same one). *Needed for Security:* 1. One of the two keys must be kept secret. 2. It must be impossible or at least impractical to decipher a message if no other information is available. 3. Knowledge of the algorithm plus one of the keys plus samples of ciphertext must be insufficient to determine the other key.

Public-Key Requirements

- Public-Key algorithms rely on two keys where:
  - it is computationally infeasible to find decryption key knowing only algorithm & encryption key
  - it is computationally easy to en/decrypt messages when the relevant (en/decrypt) key is known

  - either of the two related keys can be used for encryption, with the other used for decryption (for some algorithms)
- these are formidable requirements which only a few algorithms have satisfied

Requirements for Public-Key Cryptography

However, they did lay out the conditions that such algorithms must fulfill [DIFF76b]:
1. It is computationally easy for a party B to generate a pair (public key PUb, private key PRb).
2. It is computationally easy for a sender A, knowing the public key and the message to be encrypted, M, to generate the corresponding ciphertext: C = E(PUb, M)
3. It is computationally easy for the receiver B to decrypt the resulting ciphertext using the private key to recover the original message: M = D(PRb, C) = D[PRb, E(PUb, M)]
4. It is computationally infeasible for an adversary, knowing the public key, PUb, to determine the private key, PRb.
5. It is computationally infeasible for an adversary, knowing the public key, PUb, and a ciphertext, C, to recover the original message,

We can add a sixth requirement that, although useful, is not necessary for all public-key applications:
6. The two keys can be applied in either order: M = D[PUb, E(PRb, M)] = D[PRb, E(PUb, M)]

One-way function

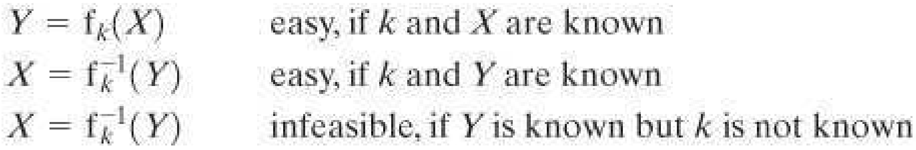
A *one-way function* is one that maps a domain into a range such that every function value has a unique inverse, with the condition that the calculation of the function is easy whereas the calculation of the inverse is infeasible:

Y = f(X) easy
X = f_1(X) infeasible

Generally, *easy* is defined to mean a problem that can be solved in polynomial time as a function of input length. Thus, if the length of the input is n bits, then the time to compute the function is proportional to n_a where a is a fixed constant. Such algorithms are said to belong to the class **P**. The term *infeasible* is a much fuzzier concept. In general, we can say a problem is infeasible if the effort to solve it grows faster than polynomial time as a function of input size. For example, if the length of the input is n bits and the time to compute the function is proportional to 2_n, the problem is considered infeasible. Unfortunately, it is difficult to determine if a particular algorithm exhibits this complexity.
Furthermore, traditional notions of computational complexity focus on the worst-case or average-case complexity of an algorithm. These measures are inadequate for cryptography, which requires that it be infeasible to invert a function for virtually all inputs, not for the worst case or even average case.

Trap-door one-way function

A *trap-door one-way function*, which is easy to calculate in one direction and infeasible to calculate in the other direction unless certain additional information is known. With the additional information the inverse can be calculated in polynomial time. We can summarize as follows: A trap-door one-way function is a family of invertible functions f*k*, such that



$Y = f_k(X)$ easy, if k and X are known

$X = f_k^{-1}(Y)$ easy, if k and Y are known

$X = f_k^{-1}(Y)$ infeasible, if Y is known but k is not known

Thus, the development of a practical public-key scheme depends on discovery of a suitable trap-door one-way function.

Security of Public Key Schemes

- like private key schemes brute force **exhaustive search** attack is always theoretically possible
- but keys used are too large (>512bits)
- security relies on a **large enough** difference in difficulty between **easy** (en/decrypt) and **hard** (cryptanalyse) problems
- more generally the **hard** problem is known, but is made hard enough to be impractical to break
- requires the use of **very large numbers**
- hence is **slow** compared to private key schemes

3.2 Introduction to Number Theory

Objectives

- Introduction to number theory.
- Fundamental Theorem of Arithmetic
- Fermats Theorem

Introduction to Number Theory - Number theory is about **integers** and their properties.

Basic principles of number theory

- divisibility,
- greatest common divisors,
- least common multiples, and
- modular arithmetic

Divisors

- say a non-zero number b **divides** a if for some m have a=mb (a,b,m all integers)
- that is b divides into a with no remainder
- denote this b|a and say that b is a **divisor** of a
- eg. all of 1,2,3,4,6,8,12,24 divide 24
- eg. 13 | 182; –5 | 30; 17 | 289; –3 | 33; 17 | 0

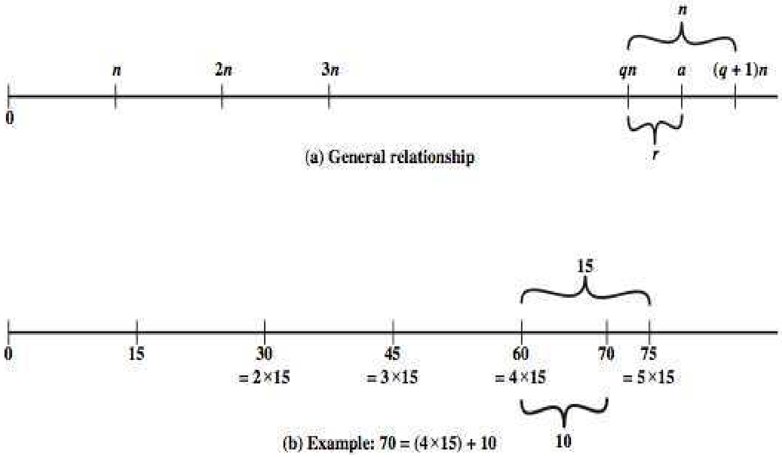
Properties of Divisibility

- If *a|1, then a = ±1.*
- *If a|b and b|a, then a = ±b.*

- *Any b /= 0 divides 0.*
- *If a | b and b | c, then a | c*
  - *e.g.* 11 | 66 and 66 | 198 x 11 | 198
- If *b|g and b|h, then b|(mg + nh)*
- *for arbitrary integers m and n*
  - *e.g. b = 7; g = 14; h = 63; m = 3; n = 2*
  - *hence 7|14 and 7|63*

Division Algorithm

- if divide a by n get integer quotient *q* and integer remainder *r* such that:
  - *a = qn + r* where *0 <= r < n; q = floor(a/n)*
- remainder *r* often referred to as a **residue**



(a) General relationship

(b) Example: 70 = (4×15) + 10

Greatest Common Divisor (GCD)- a common problem in number theory

- GCD (a,b) of a and b is the largest integer that divides evenly into both a and b
  - eg GCD(60,24) = 12
- define gcd(0, 0) = 0
- often want **no common factors** (except 1) define such numbers as **relatively prime**
  - eg GCD(8,15) = 1
  - hence 8 & 15 are relatively prime

Example GCD(1970,1066)

1970 = 1 x 1066 + 904	gcd(1066, 904)
1066 = 1 x 904 + 162	gcd(904, 162)
904 = 5 x 162 + 94	gcd(162, 94)
162 = 1 x 94 + 68	gcd(94, 68)
94 = 1 x 68 + 26	gcd(68, 26)
68 = 2 x 26 + 16	gcd(26, 16)
26 = 1 x 16 + 10	gcd(16, 10)
16 = 1 x 10 + 6	gcd(10, 6)
10 = 1 x 6 + 4	gcd(6, 4)
6 = 1 x 4 + 2	gcd(4, 2)
4 = 2 x 2 + 0	gcd(2, 0)

- **Primes**

  A positive integer p greater than 1 is called prime if the only positive factors of p are 1 and p.
- A positive integer that is greater than 1 and is not prime is called composite.

  The fundamental theorem of arithmetic:

Every positive integer can be written **uniquely** as the **product of primes**, where the prime factors are written in order of increasing size.

Prime Factorisation

- To **factor** a number n is to write it as a product of other numbers: n=a × b × c
- The **prime factorisation** of a number n is when its written as a product of primes
  - eg. 91=7×13 ; 3600=$2^4 \times 3^2 \times 5^2$
  -

$$a = \prod_{p \in \mathrm{P}} p^{a_p}$$

15 =3·5

48 =2·2·2·2·3 = $2^4 \cdot 3$

17 =17

100 =2·2·5·5 = $2^2 \cdot 5^2$

512 =2·2·2·2·2·2·2·2·2 = 2^9

Relatively Prime Numbers

- Two numbers **a, b** are **relatively prime** if have **no common divisors** apart from 1.
  - eg. 8 & 15 are relatively prime since factors of 8 are 1,2,4,8
  - and of 15 are 1,3,5,15
  - and 1 is the only common factor

Fermat's Theorem

- If *p* is prime and *a* is a positive integer not divisible by *p*

a^{p-1} º 1 (mod *p)*

where p is prime and **gcd(a,p)=1**

a^p º *a* (mod *p*)

- useful in public key and primality testing.

Fermat's Little Theorem

- If *p* is prime and *a* is an integer not divisible by *p*, then . . .
- a^{p-1} 1 (mod *p*).
- And for every integer *a*
- a^p *a* (mod *p*).
- This theorem is useful in public key (RSA) and primality testing.

Euler Totient Function ø(n)

- When doing arithmetic modulo n
- **Complete set of residues** is: 0..n-1.
- **Reduced set of residues** is those numbers (residues) which are relatively prime to n
  - eg for n=10,
  - complete set of residues is {0,1,2,3,4,5,6,7,8,9}
  - reduced set of residues is {1,3,7,9}

Number of elements in reduced set of residues is called the Euler Totient Function ø(n)

- To compute ø(n) , need to count number of elements to be excluded.
- in general need prime factorization, but
  - **For p, ø(p) = p-1 where p is prime.**
  - **For p.q ø(p.q) = (p-1)(q-1)** where **p & q are prime.**
- eg.
  - **ø(37) = 36**
  - **ø(21) = (3–1)×(7–1) = 2×6 = 12**
- Euler Totient Function ø(n)

- ø (*n*) = how many numbers there are between 1 and *n*-1 that are relatively prime to *n*.
- ø (4) = 2 (1, 3 are relatively prime to 4)
- ø (5) = 4 (1, 2, 3, 4 are relatively prime to 5)
- ø (6) = 2 (1, 5 are relatively prime to 6)
- ø (7) = 6 (1, 2, 3, 4, 5, 6 are relatively prime to 7)

- Let's test the theorem:
- If *a* = 5 and *p* = 6
- Then (6) = (2-1) * (3-1) = 2
- So, $5^{(6)}$ = 25 and 25 = 24+1 = 6*4+1
- => 25 = 1(mod 6) OR 25 % 6 = 1
- It also follows that $a^{(p)+1}$ *a*(mod *p*) so that *p* does not necessarily need to be relatively prime to *a*.

Euler's Theorem

- For every *a* and *n* that are relatively prime

a ø (*n*) = 1 mod n
(a ø (*n*) +1 $^{=}$ a mod n)

Euclidean Algorithm

- **an efficient way to find the GCD(a,b)**
- uses theorem that:

- GCD(a,b) = GCD(b, a mod b)

➢ Euclidean Algorithm to compute GCD(a,b) is:

```
Euclid(a,b)
        if (b=0) then return a;
        else return Euclid(b, a mod b);
```

Extended Euclidean Algorithm

➢ calculates not only GCD but x & y:

$$ax + by = d = \gcd(a, b)$$

➢ useful for later crypto computations

➢ follow sequence of divisions for GCD but assume at each step i, can find x &y: r = ax + by

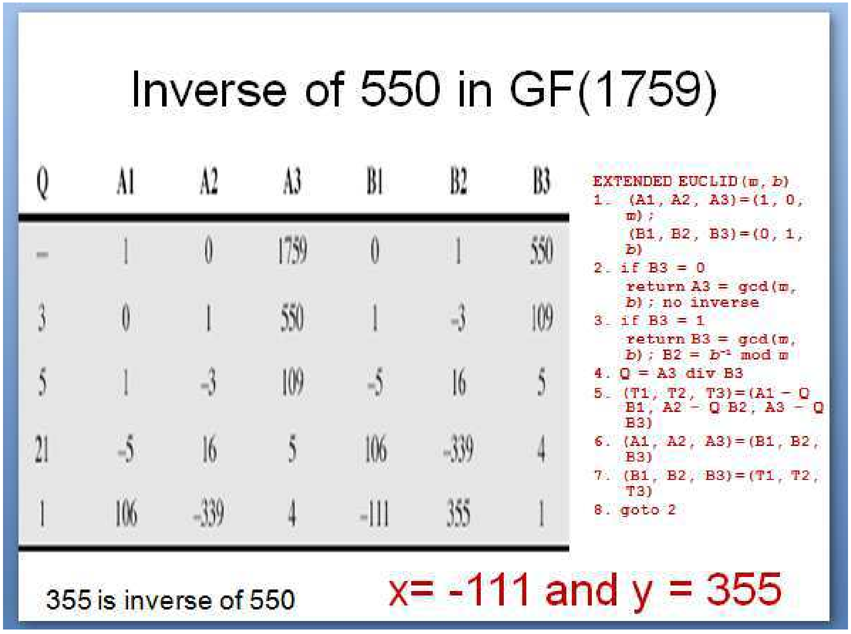
➢ at end find GCD value and also x & y

➢ if GCD(a,b)=1 these values are inverses

Finding Inverses

```
EXTENDED EUCLID(m, b)
1. (A1, A2, A3)=(1, 0, m);
        (B1, B2, B3)=(0, 1, b)
2. if B3 = 0
        return A3 = gcd(m, b); no inverse
3. if B3 = 1
        return B3 = gcd(m, b); B2 = $b^{-1}$ mod m
4. Q = A3 div B3
5. (T1, T2, T3)=(A1 – Q B1, A2 – Q B2, A3 – Q B3)
6. (A1, A2, A3)=(B1, B2, B3)
7. (B1, B2, B3)=(T1, T2, T3)
8. goto 2
```

Inverse of 550 in GF(1759)



Inverse of 550 in GF(1759)

Q	A1	A2	A3	B1	B2	B3
–	1	0	1759	0	1	550
3	0	1	550	1	-3	109
5	1	-3	109	-5	16	5
21	-5	16	5	106	-339	4
1	106	-339	4	-111	355	1

```
EXTENDED EUCLID(m, b)
1. (A1, A2, A3)=(1, 0, m);
   (B1, B2, B3)=(0, 1, b)
2. if B3 = 0
   return A3 = gcd(m, b); no inverse
3. if B3 = 1
   return B3 = gcd(m, b); B2 = b⁻¹ mod m
4. Q = A3 div B3
5. (T1, T2, T3)=(A1 – Q B1, A2 – Q B2, A3 – Q B3)
6. (A1, A2, A3)=(B1, B2, B3)
7. (B1, B2, B3)=(T1, T2, T3)
8. goto 2
```

355 is inverse of 550

x= -111 and y = 355

Modular Arithmetic

- define **modulo operator** "a mod n" to be remainder when a is divided by n
  - where integer *n* is called the **modulus**
- *b* is called a **residue** of *a* mod *n*
  - since with integers can always write: a = qn + b
  - usually chose smallest positive remainder as residue
    - ie. 0 <= b <= n-1
  - process is known as **modulo reduction**
    - eg. -12 mod 7 = -5 mod 7 = 2 mod 7 = 9 mod 7
- *a* & *b* are **congruent** if: a mod n = b mod n
  - when divided by *n,* a & b have same remainder
  - eg. 100 = 34 mod 11

Modular Arithmetic Operations

- can perform arithmetic with residues
- uses a finite number of values, and loops back from either end

Z_n = {0, 1, . . . , *(n – 1)}*

- modular arithmetic is when do addition & multiplication and modulo reduce answer
- can do reduction at any point, ie
  - a+b mod n = [a mod n + b mod n] mod n

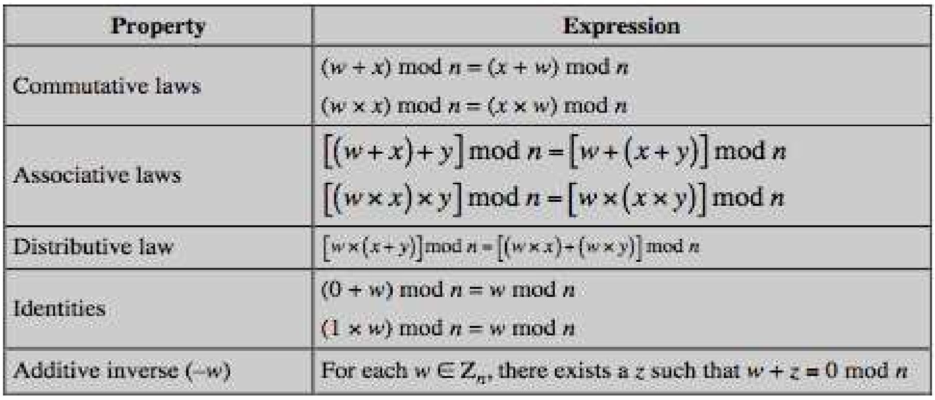
Modular Arithmetic Operations

1. [(a mod n) + (b mod n)] mod n = (a + b) mod n
2. [(a mod n) – (b mod n)] mod n = (a – b) mod n
3. [(a mod n) x (b mod n)] mod n = (a x b) mod n

e.g.

[(11 mod 8) + (15 mod 8)] mod 8 = 10 mod 8 = 2 (11 + 15) mod 8 = 26 mod 8 = 2
[(11 mod 8) – (15 mod 8)] mod 8 = –4 mod 8 = 4 (11 – 15) mod 8 = –4 mod 8 = 4
[(11 mod 8) x (15 mod 8)] mod 8 = 21 mod 8 = 5 (11 x 15) mod 8 = 165 mod 8 = 5

Modular Arithmetic Properties



Property	Expression
Commutative laws	$(w + x) \bmod n = (x + w) \bmod n$ $(w \times x) \bmod n = (x \times w) \bmod n$
Associative laws	$[(w+x)+y] \bmod n = [w+(x+y)] \bmod n$ $[(w\times x)\times y] \bmod n = [w\times(x\times y)] \bmod n$
Distributive law	$[w\times(x+y)] \bmod n = [(w\times x)+(w\times y)] \bmod n$
Identities	$(0 + w) \bmod n = w \bmod n$ $(1 \times w) \bmod n = w \bmod n$
Additive inverse $(-w)$	For each $w \in Z_n$, there exists a z such that $w + z = 0 \bmod n$

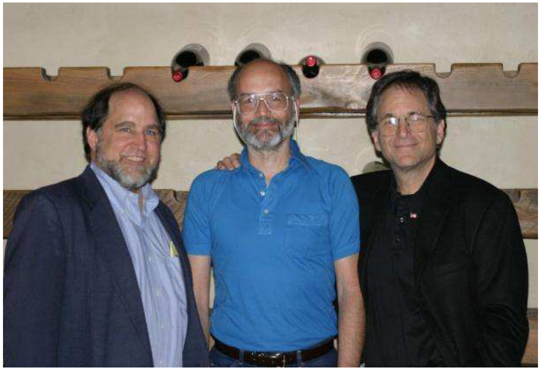
3.4 RSA Algorithm

Objectives

- To Explain &Illustrate RSA algorithm

The RSA scheme is a block cipher in which the plaintext and ciphertext are integers between 0 and *n*- 1 for some *n*. A typical size for *n* is 1024 bits, or 309 decimal digits. That is, *n* is less than 21024.

- By Ron Rivest, Adi Shamir & Len Adleman of MIT in 1977 .



RSA

- **best known & widely used public-key scheme** .
- **based on exponentiation** in a finite field over integers modulo a prime .
- **uses large integers** (eg. 1024 bits).
- **security** due to cost of factoring large numbers

RSA Scheme

- Block cipher scheme.
- Plain text encrypted in blocks.
- Each block having a binary value less than some number n.
- Plain text and cipher text are integers between 0 and n-1 for some n.
- Typical size of n is 1024 bits.(309 decimal digits).

RSA Encryption /Decryption

$C = M^e$ mod n

$M = C^d$ mod n

Using the encryption key ,e And decryption key, d .

RSA Key Setup

Each user generates a public/private key pair by:

Selecting **two** large **primes** at random – **p, q .**

computing their system modulus

n = p.q

note ø(n)=(p-1)(q-1)

Selecting at random **the encryption key ,'e'**

where **1<e<ø(n), gcd(e,ø(n))=1**

solve following equation to find **decryption key, d**

e.d mod ø(n) =1

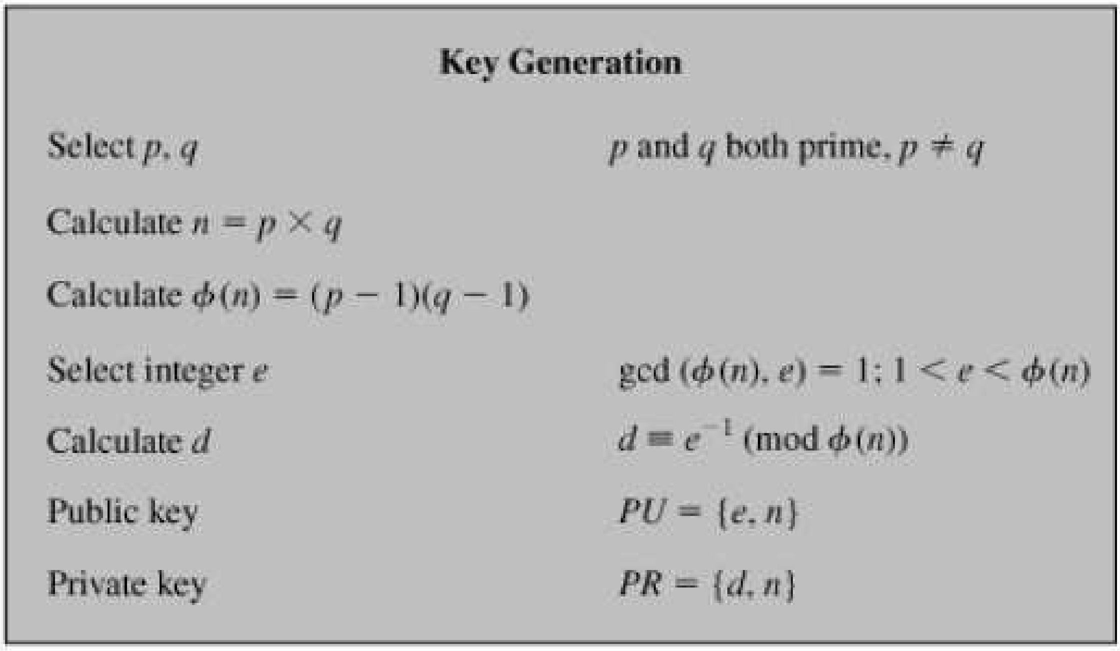
e.d ≡1 mod ø(n) and 0≤d≤n

or d=e^{-1} mod ø(n)
publish their public encryption key: KU={e,n}
keep secret private decryption key: KR={d,n}

RSA Use

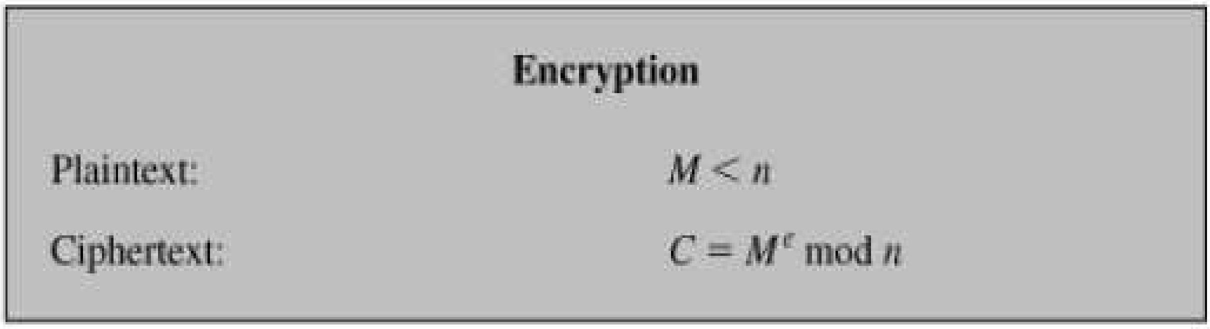
■ To encrypt a message M the sender:
obtains **public key of recipient KU={e,n}**
☐ **computes: $C = M^e \bmod n$,**
where $0 \leq M < n$

■ To decrypt the ciphertext C the owner:
uses their private key KR={d,n}
☐ **computes: $M = C^d \bmod n$**



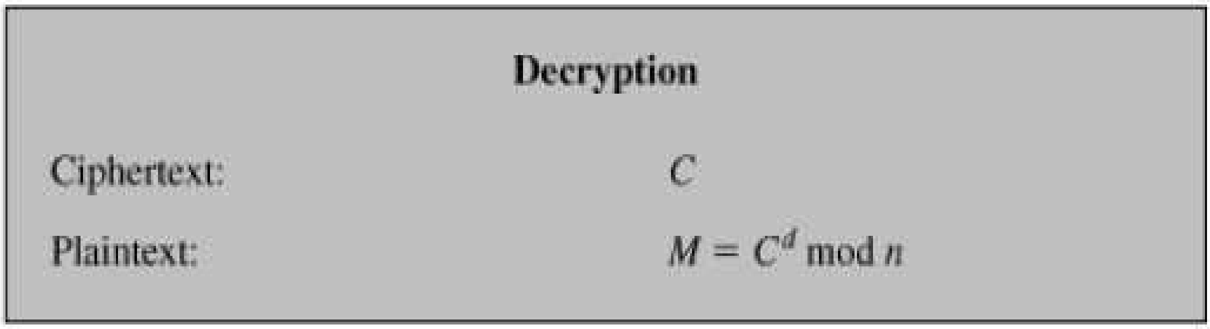
Key Generation

Select p, q	p and q both prime, $p \neq q$
Calculate $n = p \times q$	
Calculate $\phi(n) = (p-1)(q-1)$	
Select integer e	$\gcd(\phi(n), e) = 1; 1 < e < \phi(n)$
Calculate d	$d \equiv e^{-1} \pmod{\phi(n)}$
Public key	$PU = \{e, n\}$
Private key	$PR = \{d, n\}$



Encryption

Plaintext:	$M < n$
Ciphertext:	$C = M^e \bmod n$



Decryption

Ciphertext:	C
Plaintext:	$M = C^d \bmod n$

RSA Example (1)

1. Select primes: ***p*=3 & *q*=11**
2. Compute ***n* = *pq* =3×11=33**
3. Compute **ø(*n*)=(*p*–1)(*q*-1)=2×10=20**
4. Select e *:* **gcd(e,20)=1;e< ø(*n*)**choose ***e*=7**
5. Determine d*:* ***de*=1 mod 20**

and *d* < 20 Value is **d=3** since 3×7=21 which gives 21 mod 20= 1

6. Publish public key **KU={7,33}**
7. Keep secret private key **KR={3,33}**

- sample RSA encryption/decryption is:
- given message M = 2 (note . 2<33)
- encryption:

$C = M^e$ mod n
$C = 2^7$ mod 33 = 29

- decryption:

$M = C^d$ mod n
$M = 29^3$ mod 33 = 2

RSA Example (2)

1. Select primes: p=17 & q=11
2. Compute $n = pq$ =17×11=187
3. Compute $\phi(n)=(p–1)(q-1)$=16×10=160
4. Select e *:* gcd(e,160)=1; choose e=7
5. Determine d*:* de=1 mod 160 and d < 160 Value is d=23 since 23×7=161= 10×160+1
6. Publish public key KU={7,187}
7. Keep secret private key KR={23,17,11}

- sample RSA encryption/decryption is:
- given message M = 88 (nb. 88<187)
- encryption:

$C = 88^7$ mod 187 = 11

- decryption:

$M = 11^{23}$ mod 187 = 88

Example of RSA Algorithm

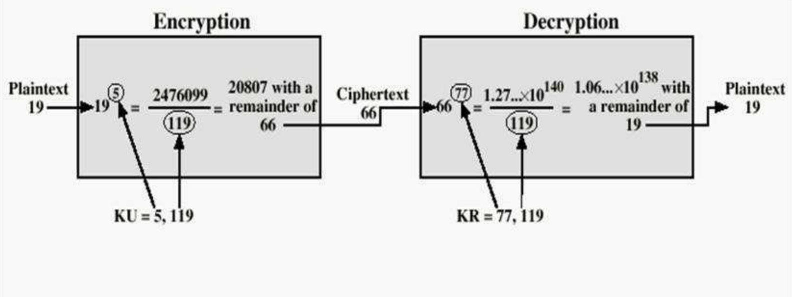


Figure 3.9 Example of RSA Algorithm

RSA Security

- 4 possible Approaches to attacking RSA:
    - Brute force key search (infeasible given size of numbers)

- Mathematical attacks (based on difficulty of computing ø(N), by factoring modulus N)
- Timing attacks (on running time of decryption)
- Chosen Cipher attacks.

Procedure for picking a Prime Number

In summary, the procedure for picking a prime number is as follows.
1. Pick an odd integer *n* at random (e.g., using a pseudorandom number generator).
2. Pick an integer *a* < *n* at random.
3. Perform the probabilistic primality test, such as Miller-Rabin, with *a* as a parameter. If *n* fails the
test, reject the value *n* and go to step 1.

4. If *n* has passed a sufficient number of tests, accept *n*; otherwise, go to step 2.

4.5 Key Management

Public-key encryption helps address key distribution problems

- have two aspects of this:

1. **Distribution of public keys.**
2. **Use of public-key encryption to distribute secret keys.**

Distribution of Public Keys can be considered as using one of:

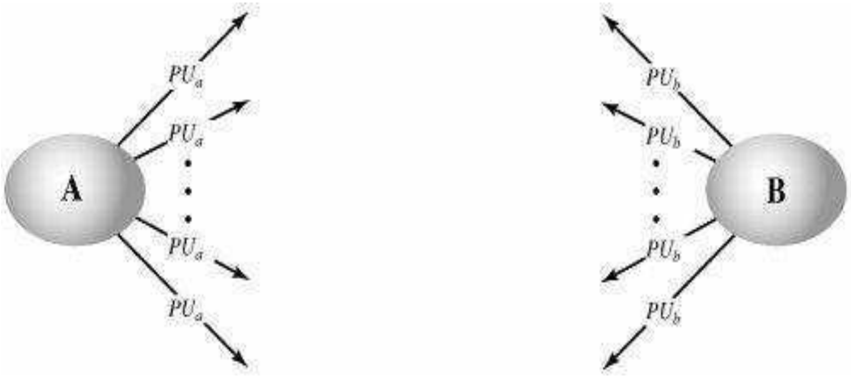
1. **Public announcement**
2. **Publicly available directory**
3. **Public-key authority**
4. **Public-key certificates**

Public Announcement

Users distribute public keys to recipients or broadcast to community at large.

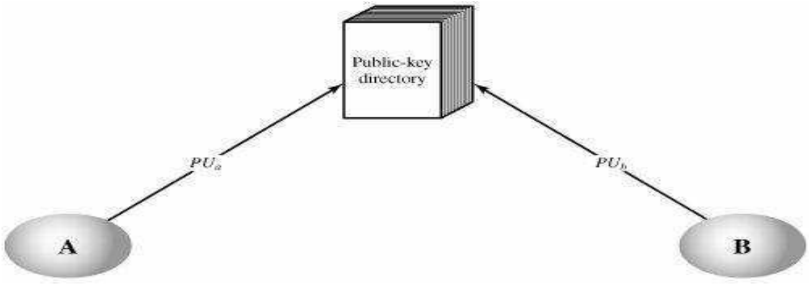
– eg. append Public keys to email messages or post to news groups or email list

- **major weakness is forgery**
  – anyone can create a key claiming to be someone else and broadcast it
  – until forgery is discovered can masquerade as claimed user



Publicly Available Directory

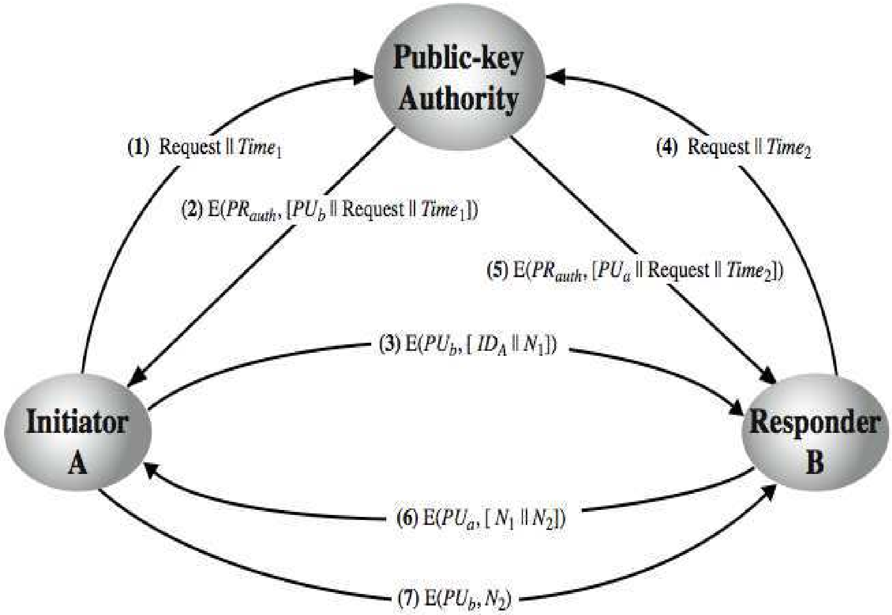
- Can obtain greater security by registering keys with a public directory.
- Directory must be trusted with properties:
  - contains {name, public-key} entries.
  - Participants may register securely with directory.
  - participants can replace key at any time.
  - directory is periodically published.
  - directory can be accessed electronically.
- still vulnerable to tampering or forgery



  - Participants may register securely with directory.
  - Contains {name, public-key} entries
- Then users interact with directory to obtain any desired public key securely
- improve security by tightening control over distribution of keys from directory
- has properties of directoryand requires users to know public key for the directory.
- then users interact with directory to obtain any desired public key securely
  - does require real-time access to directory when keys are needed.

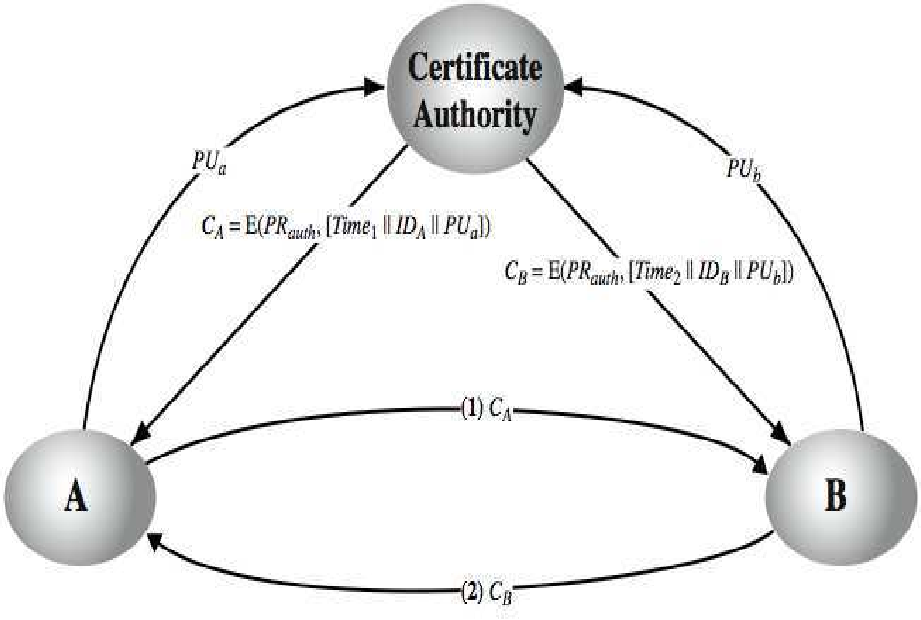
Public-Key Authority

- Improve security by tightening control over distribution of keys from directory.
- has properties of directory.
- and requires users to know public key for the directory.
- then users interact with directory to obtain any desired public key securely
  - does require real-time access to directory when keys are needed
  - may be vulnerable to tampering


Public-key Authority
(1) Request || $Time_1$
(2) E(PR_{auth}, [PU_b || Request || $Time_1$])
(4) Request || $Time_2$
(5) E(PR_{auth}, [PU_a || Request || $Time_2$])
(3) E(PU_b, [ID_A || N_1])
Initiator A
Responder B
(6) E(PU_a, [N_1 || N_2])
(7) E(PU_b, N_2)

Public-Key Certificates

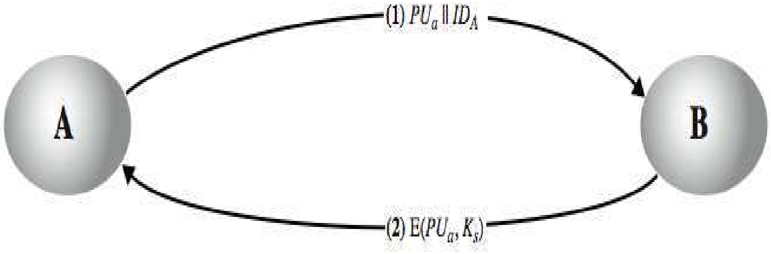
- Certificates allow key exchange without real-time access to public-key authority
- A certificate binds identity to public key
  - usually with other info such as period of validity, rights of use etc
- with all contents signed by a trusted Public-Key or Certificate Authority (CA)
- can be verified by anyone who knows the public-key authorities public-key


Certificate Authority
PU_a
PU_b
C_A = E(PR_{auth}, [$Time_1$ || ID_A || PU_a])
C_B = E(PR_{auth}, [$Time_2$ || ID_B || PU_b])
(1) C_A
A
B
(2) C_B

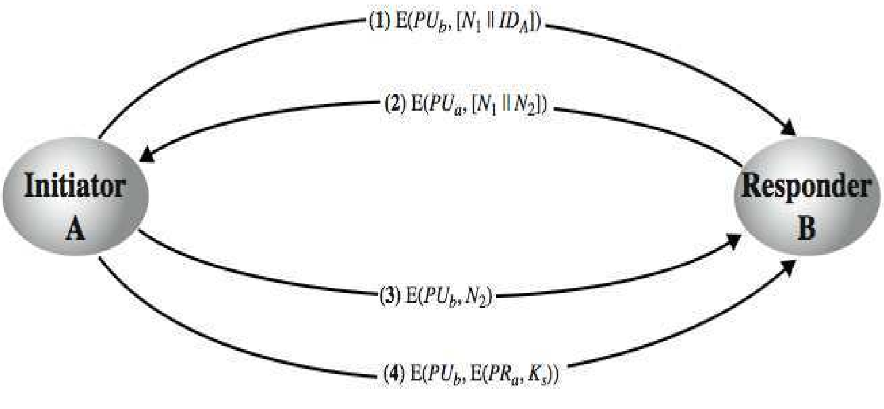
Use of public-key encryption to distribute secret keys

- use previous methods to obtain public-key
- can use for secrecy or authentication
- but public-key algorithms are slow
- so usually want to use private-key encryption to protect message contents
- hence need a session key
- have several alternatives for negotiating a suitable session

Simple Secret Key Distribution



Secret Key Distribution with Confidentiality and Authentication

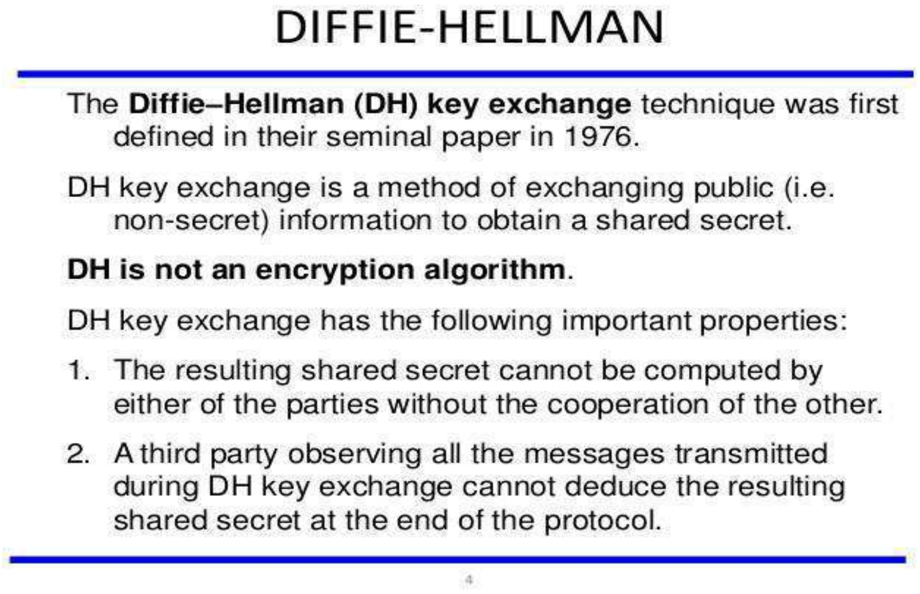

(1) $E(PU_b, [N_1 \| ID_A])$
(2) $E(PU_a, [N_1 \| N_2])$
Initiator A
Responder B
(3) $E(PU_b, N_2)$
(4) $E(PU_b, E(PR_a, K_s))$

4.6 Diffie-Hellman Key Exchange

- By Whitfield Diffie and Martin Hellman in 19766 along with the exposition of public key concepts

The purpose of the algorithm is to enable two users to securely exchange a key that can then be used for subsequent encryption of messages. The algorithm itself is limited to the exchange of secret values.

- Developed in 1976 and published in “New Directions in Cryptography.”
- The protocol allows two users to exchange secret key over an insecure medium without any prior secrets.
  - used in a number of commercial products



DIFFIE-HELLMAN

The **Diffie–Hellman (DH) key exchange** technique was first defined in their seminal paper in 1976.

DH key exchange is a method of exchanging public (i.e. non-secret) information to obtain a shared secret.

DH is not an encryption algorithm.

DH key exchange has the following important properties:

1. The resulting shared secret cannot be computed by either of the parties without the cooperation of the other.
2. A third party observing all the messages transmitted during DH key exchange cannot deduce the resulting shared secret at the end of the protocol.

4

For this scheme, there are two publicly known numbers: a prime number q and an integer that is a primitive root of q. Suppose the users A and B wish to exchange a key. User A selects a random integer $X_A < q$ and computes $Y_A = \alpha^{X_A} \bmod q$. Similarly, user B independently selects a random integer $X_A < q$ and computes $Y_B = \alpha^{X_B} \bmod q$.
Each side keeps the X value private and makes the Y value available publicly to the other side. User A computes the key as $K = (Y_B)^{X_A} \bmod q$ and user B computes the key as $K = (Y_A)^{X_B} \bmod q$. These two calculations produce identical results:

$K = (Y_B)^{X_A} \bmod q$

$= (\alpha^{X_B} \bmod q)^{X_A} \bmod q$

$= (\alpha^{X_B})^{X_A} \bmod q$ by the rules of modular arithmetic
$= (\alpha^{X_B X_A} \bmod q$
$= (\alpha^{X_A})^{X_B} \bmod q$
$== (\alpha^{X_A} \bmod q)^{X_B} \bmod q$
$= (Y_A)^{X_B} \bmod q$

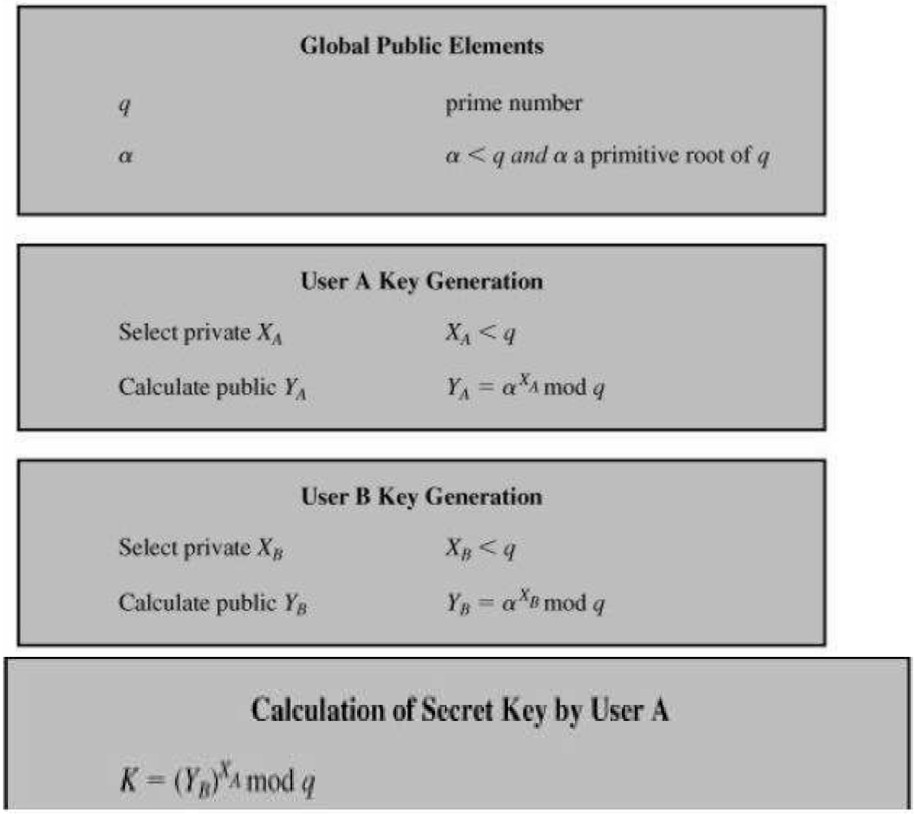
Diffie-Hellman Setup

- All users agree on global parameters:
    - **large prime integer or polynomial q**
    - **α a primitive root of q**

- **Each user (eg. A) generates their key**
    - **chooses a secret key (number): $x_A < q$**
    - **compute their public key:**
    - **$y_A = \alpha^{xA}$ mod q**
- **each user makes public that key y_A**
- **Compute Secret Key as**
  $K_{A=}\ y_B^{xA}$ mod q

- **User (eg. B) generates their key**
    - **chooses a secret key (number): $x_B < q$**
    - **compute their public key:**
    - **$y_B = \alpha^{xB}$ mod q**
- **each user makes public that key y_B**
- **Secret Key $K_{B=}\ y_A^{xB}$ mod q**

Figure 10.7. The Diffie-Hellman Key Exchange Algorithm



Global Public Elements

q	prime number
α	$\alpha < q$ *and* α a primitive root of q

User A Key Generation

Select private X_A	$X_A < q$
Calculate public Y_A	$Y_A = \alpha^{X_A} \bmod q$

User B Key Generation

Select private X_B	$X_B < q$
Calculate public Y_B	$Y_B = \alpha^{X_B} \bmod q$

Calculation of Secret Key by User A

$K = (Y_B)^{X_A} \bmod q$

Calculation of Secret Key by User B

$K = (Y_A)^{X_B} \bmod q$

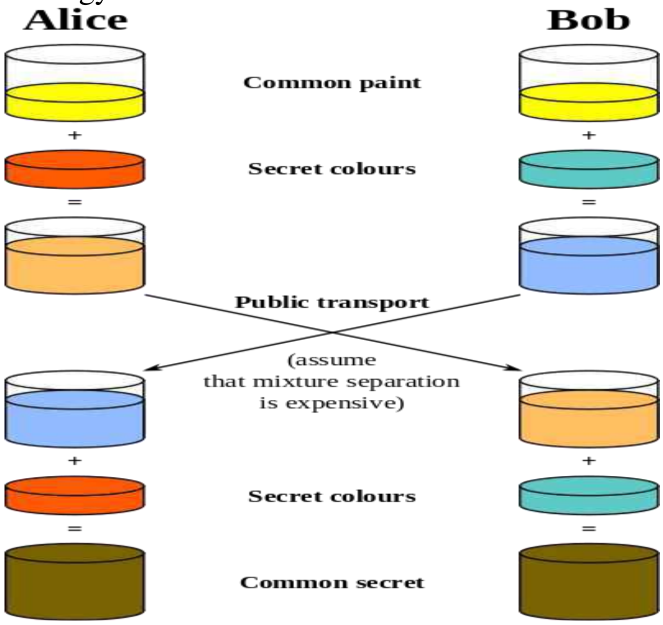
Diffie-Hellman Key Exchange

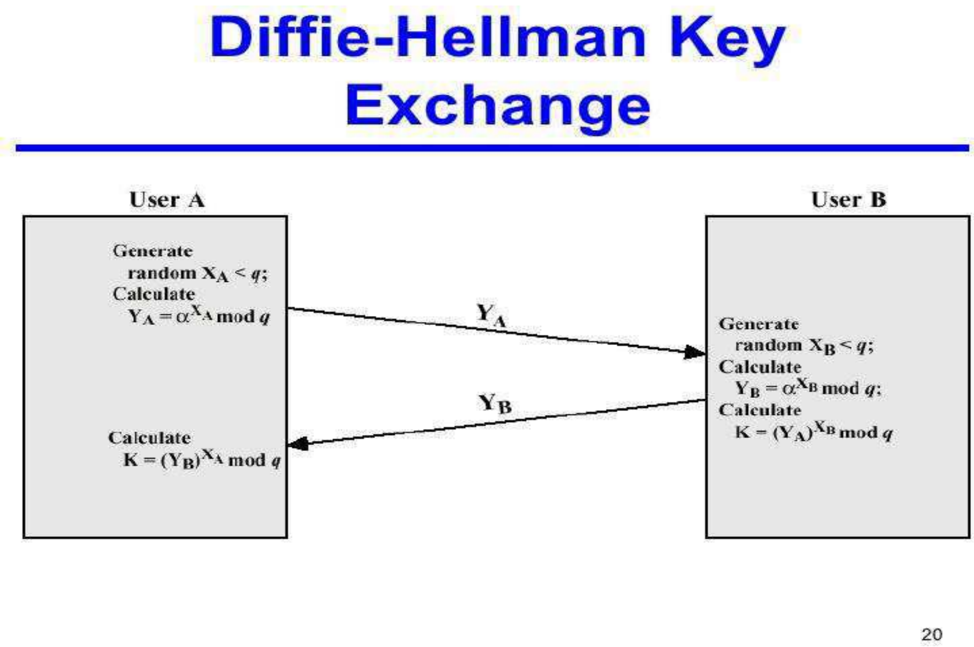
- shared session key for users A & B is K_{AB}:

$K_{AB} = \alpha^{xA.xB}$ mod q
$= y_A^{xB}$ mod q (which B can compute)
$= y_B^{xA}$ mod q (which A can compute)

- K_{AB} is used as session key in private-key encryption scheme between Alice and Bob
- if Alice and Bob subsequently communicate, they will have the same key as before, unless they choose new public-keys attacker needs an x, must solve discrete log

Analogy





Diffie-Hellman Key Exchange

User A

Generate random $X_A < q$;
Calculate $Y_A = \alpha^{X_A} \bmod q$

Calculate $K = (Y_B)^{X_A} \bmod q$

Y_A

Y_B

User B

Generate random $X_B < q$;
Calculate $Y_B = \alpha^{X_B} \bmod q$;
Calculate $K = (Y_A)^{X_B} \bmod q$

20

Diffie-Hellman Example

- users Alice & Bob who wish to swap keys:
- agree on prime q=353 and α=3
- select random secret keys:
  – A chooses x_A=97, B chooses x_B=233
- compute public keys:
  – $y_A=3^{97} \bmod 353 = 40$ (Alice)
  – $y_B=3^{233} \bmod 353 = 248$ (Bob)
- compute shared session key as:

$K_{AB}= y_B^{xA} \bmod 353 = 248^{97} \bmod 353= 160$ (Alice)
$K_{AB}= y_A^{xB} \bmod 353 = 40^{233} \bmod 353= 160$ (Bob)

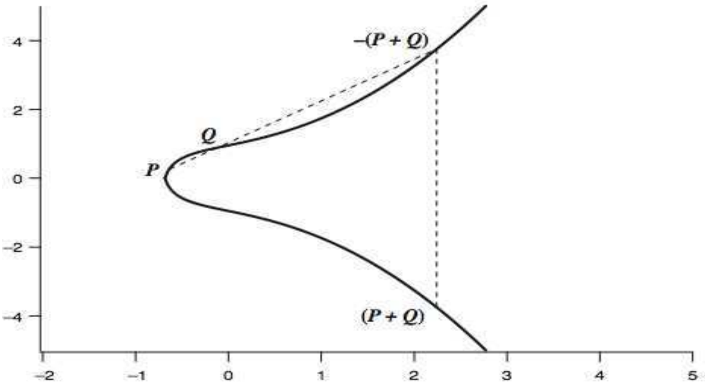
3.7 Elliptic Curve Cryptography

- majority of public-key crypto (RSA, D-H) use either integer or polynomial arithmetic with very large numbers/polynomials
- imposes a significant load in storing and processing keys and messages
- an alternative is to use elliptic curves
- offers same security with smaller bit sizes
- newer, but not as well analysed

Real Elliptic Curves

- ➢ an elliptic curve is defined by an equation in two variables x & y, with coefficients
- ➢ consider a cubic elliptic curve of form
  - ● $y^2 = x^3 + ax + b$
  - ● where x,y,a,b are all real numbers

  - also define zero point O
- consider set of points E(a,b) that satisfy
- have addition operation for elliptic curve
  - geometrically sum of P+Q is reflection of the intersection R



(b) $y^2 = x^3 + x + 1$

Finite Elliptic Curves

- Elliptic curve cryptography uses curves whose variables & coefficients are finite
- have two families commonly used:
  - prime curves $E_p(a,b)$ defined over Z_p
    - use integers modulo a prime
    - best in software
  - binary curves $E_{2m}(a,b)$ defined over $GF(2^n)$
    - use polynomials with binary coefficients
    - best in hardware

Elliptic Curve Cryptography

- ECC addition is analog of modulo multiply
- ECC repeated addition is analog of modulo exponentiation
- need "hard" problem equiv to discrete log
  - Q=kP, where Q,P belong to a prime curve
  - is "easy" to compute Q given k,P
  - but "hard" to find k given Q,P
  - known as the elliptic curve logarithm problem
- Certicom example: $E_{23}(9,17)$

ECC Diffie-Hellman

- can do key exchange analogous to D-H
- users select a suitable curve $E_q(a,b)$
- select base point $G=(x_1,y_1)$
  - with large order n s.t. nG=O
- A & B select private keys $n_A<n$, $n_B<n$
- compute public keys: $P_A=n_AG$, $P_B=n_BG$
- compute shared key: $K=n_AP_B$, $K=n_BP_A$
  - same since $K=n_An_BG$
- attacker would need to find k, hard

ECC Encryption/Decryption

- several alternatives, will consider simplest
- must first encode any message M as a point on the elliptic curve P_m
- select suitable curve & point G as in D-H
- each user chooses private key n_A<n
- and computes public key $P_A=n_AG$
- to encrypt P_m : $C_m=\{kG, P_m+kP_b\}$, k random
- decrypt C_m compute:
  - $P_m+kP_b–n_B(kG) = P_m+k(n_BG)–n_B(kG) =$

Several approaches to encryption/decryption using elliptic curves have been analyzed in the literature. This one is an analog of the ElGamal public-key encryption algorithm. The sender must first encode any message M as a point on the elliptic curve P_m (there are relatively straightforward techniques for this). Note that the ciphertext is a pair of points on the elliptic curve. The sender masks the message using random k, but also sends along a “clue” allowing the receiver who know the private-key to recover k and hence the message. For an attacker to recover the message, the attacker would have to compute k given G and kG, which is assumed hard.

ECC Security

- relies on elliptic curve logarithm problem
- fastest method is “Pollard rho method”
- compared to factoring, can use much smaller key sizes than with RSA etc
- for equivalent key lengths computations are roughly equivalent
- hence for similar security ECC offers significant computational advantages

The security of ECC depends on how difficult it is to determine k given kP and P. This is referred to as the elliptic curve logarithm problem. The fastest known technique for taking the elliptic curve logarithm is known as the Pollard rho method. Compared to factoring integers or polynomials, can use much smaller numbers for equivalent levels of security.

Module 4

- **MESSAGE AUTHENTICATION**
  - **Message authentication** is a mechanism or service used to verify the integrity of a msg . It assures that data received are exactly as sent by and that the identity of the sender is valid
    - Message authentication may also verify sequencing and timeliness
  - **Digital signature**
    - An authentication technique that also includes measures to counter repudiation by either source or destination

- **AUTHENTICATION REQUIREMENTS**
  - While communicating across a network ,following attacks can be identified:

1. **Disclosure** : Release of msg contents to any person or process not processing the appropriate cryptographic key
2. **Traffic Analysis**: Discovery of pattern of traffic between parties.

   Connection oriented->frequency and duration of connection

   Connection Oriented and Connectionless->number and length of msg's determined

3. **Masquerade msg:** Insertion of msg's into the network from a fraudulent source
4. **Content Modification**: Changes to the content of a msg; including insertion , deletion, transposition, modification
5. **Sequence Modification:** Any modification to a sequence of msg's between parties
6. **Timing Modification**: Delay or replay of msg's.

Connection oriented->entire sequence
Connectionless->individual msg is replayed

7. **Source Repudiation:** Denial of transmission of msg's by source
8. **Destination repudiation: Denial** of Receipt of msg by destination

- Measures to deal with first two attacks:
  - In the realm of message *confidentiality*, and are addressed with *encryption*
- Measures to deal with items 3 to 6
  - Message *authentication*
- Measures to deal with items 7
  - *Digital signature*

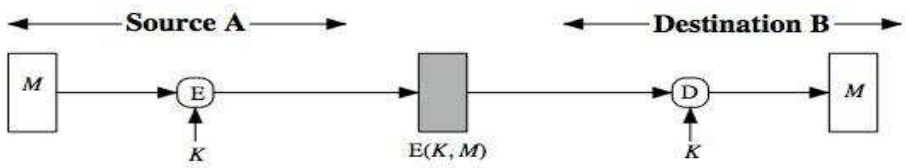
AUTHENTICATION FUNCTIONS

- Msg authentication or digital signature has two level of functionality:
  - ✓ Lower level : Some sort of function that produces an authenticator; value used to authenticate a msg
  - ✓ Higher Level : Enables a receiver to verify the authenticity of msg
- Value used to authenticate a msg can be grouped into 3 classes:
  - ✓ **Message Encryption :** Ciphertext of the entire msg serves as the authenticator
  - ✓ **Message authentication code(MAC):**Function of the msg and a secret key produces a fixed length value that serves as the authenticator
  - ✓ **Hash Function :** Function that maps a msg of any length into a fixed length hash value , which serves as the authenticator

1. MESSAGE ENCRYPTION

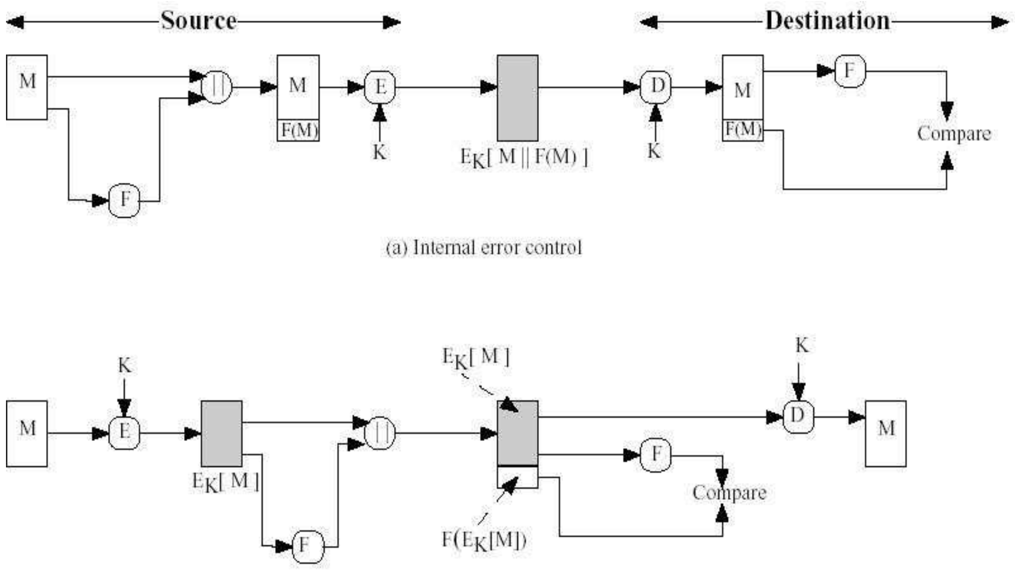
✓ Encryption itself provides an authentication

a. Symmetric Encryption:



(a) Symmetric encryption: confidentiality and authentication

- Message M transmitted from source A to destination B is encrypted using a secret key K shared by A and B
- If no other party knows the key, confidentiality is provided
- B is assured that msg was generated by A, because K is the only party that posses K
- So confidentiality and authentication
- We can add a checksum to each msg before or after encryption

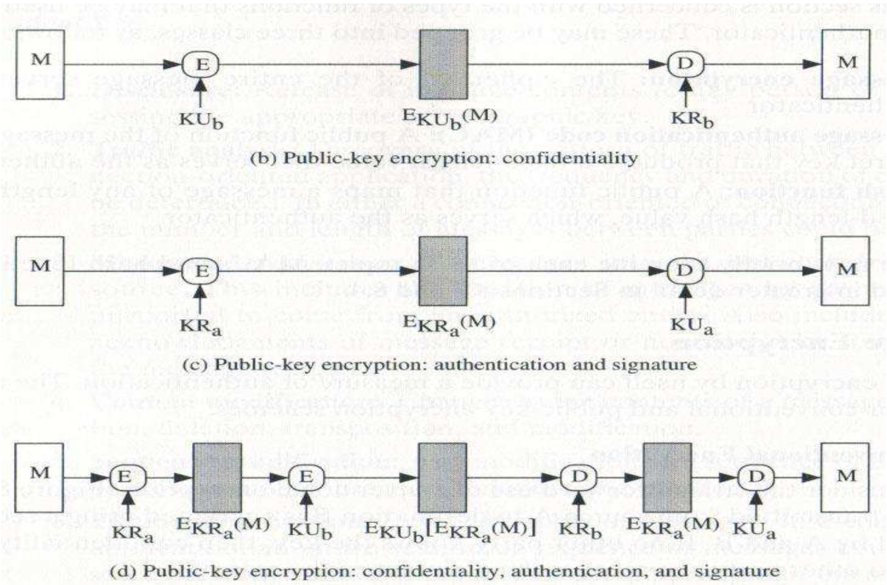


(a) Internal error control

(b) External error control

b. Public Key Encryption

- To provide both confidentiality and authentication, A can encrypt M first using its private key, which provides digital signature and then using B's public key which provides confidentiality
- **Drawback** : Public key encryption ,which is complex is used many times



(b) Public-key encryption: confidentiality

(c) Public-key encryption: authentication and signature

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

2. MESSAGE AUTHENTICATION CODE

- ✓ Authentication technique
- ✓ Use a secret key to generate a small fixed-size block of data->checksum or MAC,that is appended to the message
- ✓ Two communication parties A and B,share a common secret key K.
- ✓ When A has a msg to send to B,it calculates MAC as a function of the msg and the key: **MAC=C(K,M)**

  M->input msg

  C->MAC function

  K->shared secret key

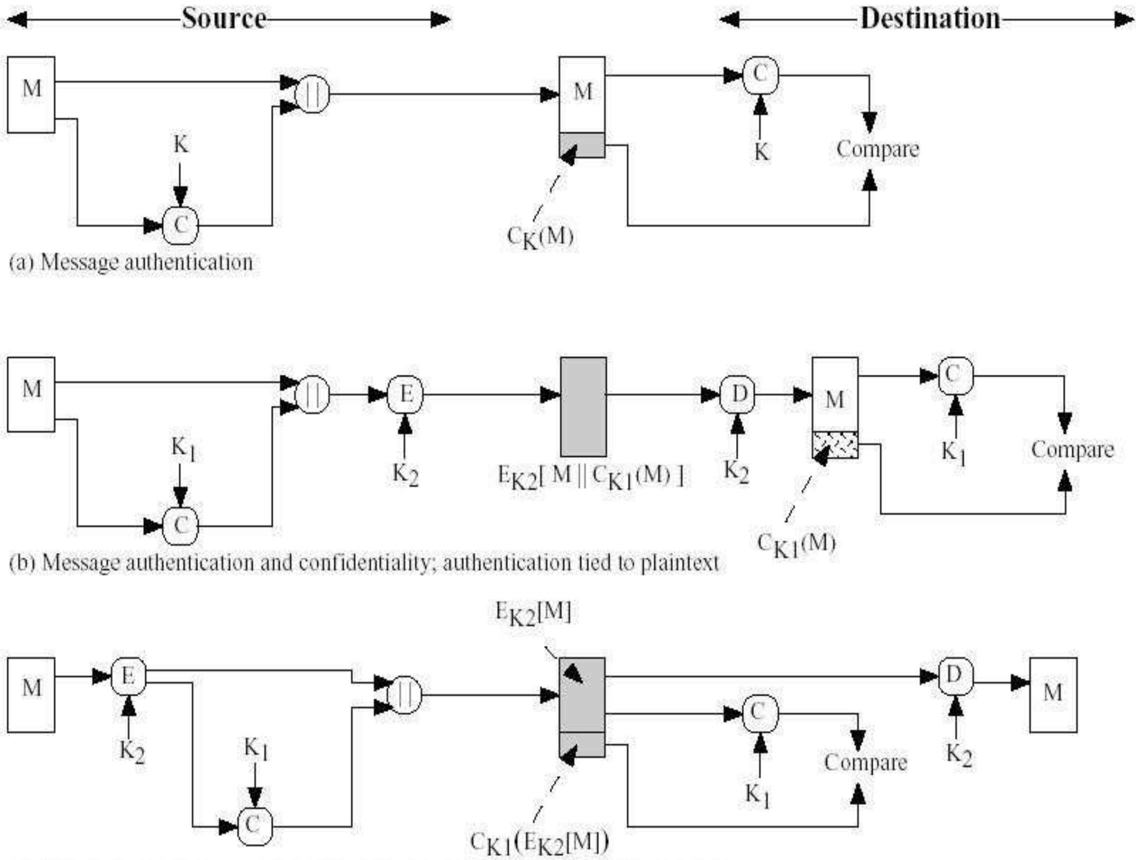
  MAC->Msg authentication code

- ✓ Msg plus MAC are transmitted to intended receiver
- ✓ Recipient performs same calculation on the received msg, using the secret key to generate a new MAC
- ✓ Received MAC is compared to the calculated MAC
- ✓ If the received MAC matches the calculated MAC, then

1. The receiver is assured that the msg has not been altered

2. The receiver is assured that the msg is from the alleged sender

3. 3.If the msg includes a sequence number, receiver can be assured of the proper sequence because an attacker canno9t successfully alter the sequence no:



(a) Message authentication

(b) Message authentication and confidentiality; authentication tied to plaintext

(c) Message authentication and confidentiality; authentication tied to ciphertext

3. HASH FUNCTION

- ✓ Hash function accepts a variable size msg as i/p and produces a fixed size output ->hash code(H(M))
- ✓ Hash code does not use a key but is a function only of the i/p msg
- ✓ Hash code is referred as message digest or hash value
- ✓ Provides error detection capability
- ✓ **Uses Of Hash Function**

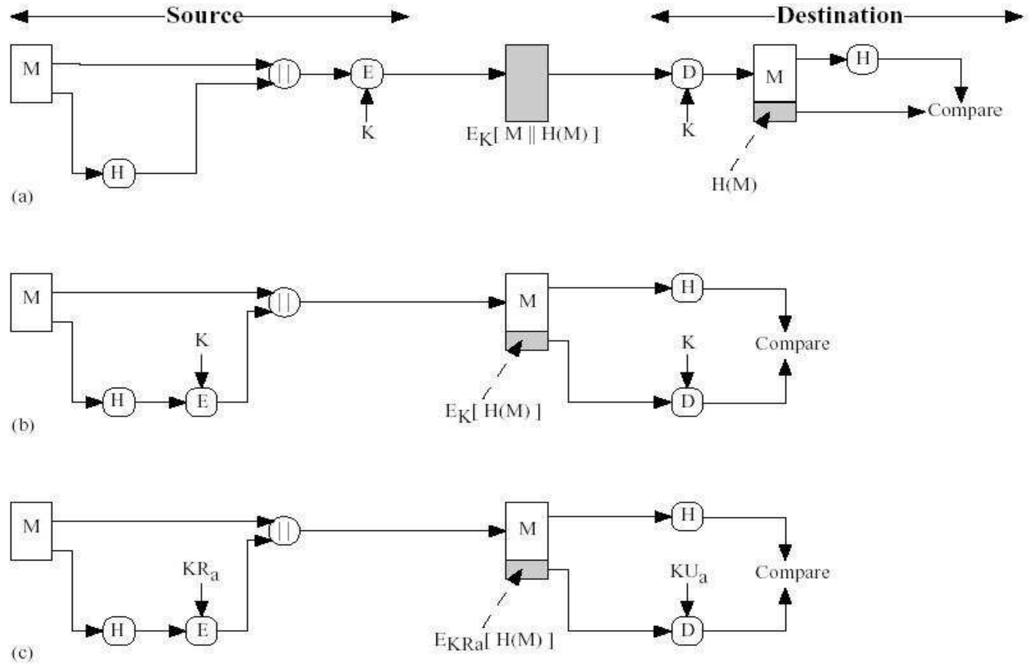


Figure 8.5 Basic Uses of Hash Function (page 1 of 2)

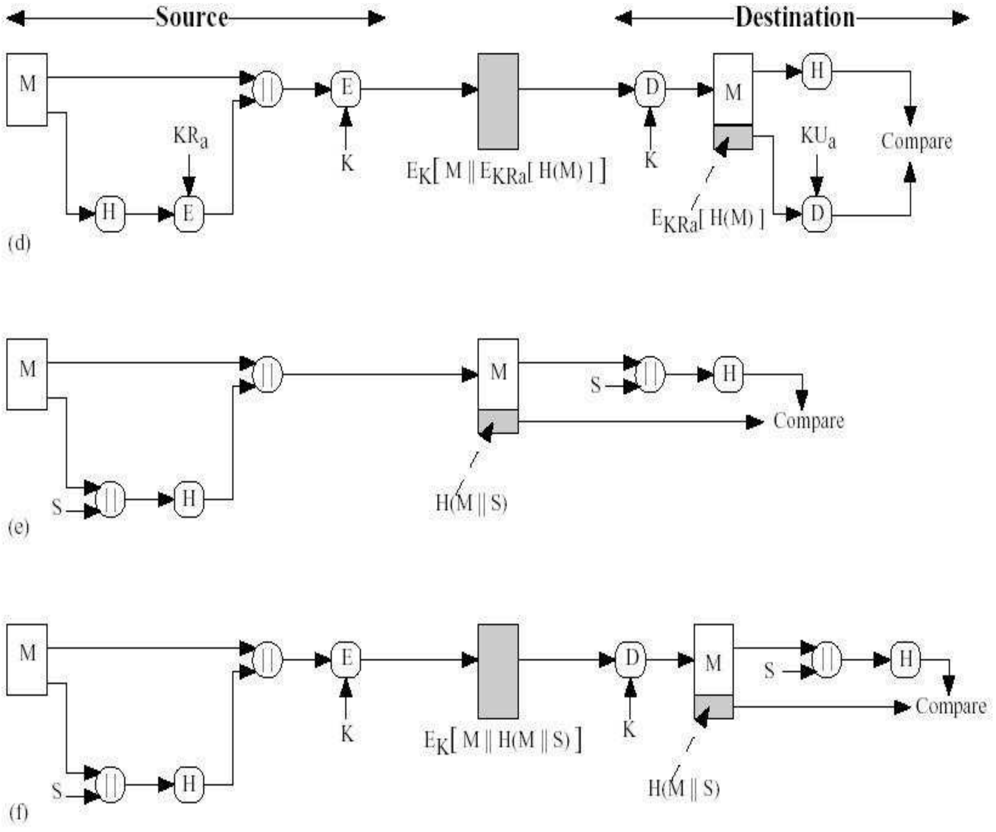


Figure 8.5 Basic Uses of Hash Function (page 2 of 2)

a) SymmetricEncryption: Authentication+Confidentiality

Here msg +hash code encrypted using symmetric key encryption

b) Hash code is encrypted using symmetric encryption

Reduces processing burden for applications that do not require confidentiality

c) Hash code is encrypted using public key encryption and use of senders private key provides authentication

d) Hash code is encrypted using public encryption. This plus msg is again encrypted using a symmetric secret key. It provides confidentiality + digital signature

e) It uses a hash function, but no encryption is performed . Here two parties share a common secret value ‘S’ .’A’ compute the hash value using this ’S’ and appends the resulting hash value to M.B uses the same S value and recompute the hash value and compare

Since the secret value itself is not sent an opponent cannot modify and generate a false msg

f) Here confidentiality is provided by encrypting the entire msg+hash code

MESSAGE AUTHENTICATION CODES

- Also known as cryptographic checksum
- Generated by a fun C of the form

MAC=C(K,M)

- M->variable length msg
- K->secret key shared only by sender & receiver
- C(K,M)->fixed length authenticator

- MAC is appended to the msg at the source
- Receiver authenticates the msg by recomputing the MAC
- MAC is irreversible, but encryption isn’t.

1. Alice and Bob share the secret K1.
2. Alice calculates $MAC_1 = C_{K1}(M)$

   Alice€Bob: $\{M, MAC_1\}$
3. Bob calculates $MAC_2 = C_{K1}(M)$

   If $MAC_2 = MAC_1$, M is sent from Alice and not altered

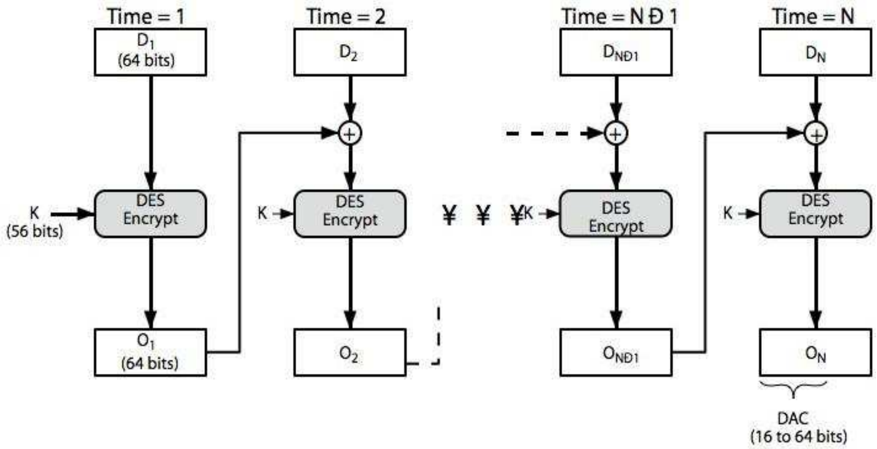
- Confidentiality can be provided by encryption with another shared key.

  Alice€Bob: $\{M, MAC_1\}_{K2}$

- **REQUIREMENTS FOR MAC'S**
  - If an opponent observes M and $C_K(M)$, it should be computationally infeasible to construct M' such that $C_K(M') = C_K(M)$.
  - $C_K(M)$ should be uniformly distributed in the sense that for randomly chosen messages, M and M', the probability that $C_K(M) = C_K(M')$ is 2^{-n}, where n is the number of bits in the MAC.
  - Let M' be equal to some known transformation on M. That is, M' = f(M). E.g. f may involve inverting one or more specific bits.
      - In that case, $Pr[C_K(M) = C_K(M')] = 2^{-n}$.

  ✓ Message Authentication Code Based on DES

  - A widely used MAC can be defined by using cipher block chaining (CBC)mode of operation of DES with an initializing vector of zero
  - Data to be authenticated are grouped into contiguous 64 bit blocks
  - Final block is padded with zeros on the right side to form full 64bit block



- ## HASH FUNCTIONS

✓ A (one-way) hash function accepts a variable-size message M as input and produces a fixed-size hash code H(M) as output (called Message Digest)

✓ Hash code provides error detection -> a change in one bit of message results in a change to the hash code

✓ **Requirements for a Hash Functions**

1. H can be applied to a block of data of any size.
2. H produces a fixed-length output.
3. It is easy to compute H(x) from any given x.
4. For any given h, computationally infeasible to find x, where H(x) = h ("one-way property")
5. For any x, computationally infeasible to find y, yÇx, H(y) = H(x) ("weak collision resistance")
6. Computationally infeasible to find any pair of (x, y) such than H(x) = H(y) ("strong collision resistance")

o **Birthday Attacks**

- might think a 64--bit hash is secure
- birthday attack works thus:

1. Source A is prepared to 'sign' a msg by appending the appropriate m-bit hash code and encrypting that hash code with A's private key
2. Opponent generates $2^{m/2}$ variations on the msg , all of which convey essentially the same meaning. Opponent prepares equal number of messages, all of which are variations on the fraudulent msg to be substituted for real one
3. Two sets of msg's are compared to find a pair of msg's that produces the same hash code. The probability of success is greater than 0.5.If no match is found, additional valid and fraudulent msgs are generated until a match is made
4. Opponent offers a valid variation to A for signature. This signature can then be attached to the fraudulent variation for transmission to the intended recipient. Because the two variations have same hash code; they will produce same signature; opponent is assured of success even though key not known

- SECURE HASH ALGORITHM

- **Hash Algorithms**
- see similarities in the evolution of hash functions & block ciphers
  - ✓ increasing power of brute-force attacks
  - ✓ leading to evolution in algorithms
  - ✓ from DES to AES in block ciphers
  - ✓ from MD4 & MD5 to SHA-1 & RIPEMD-160 in hash algorithms
- likewise tend to use common iterative structure as do block ciphers

- MD5 (MESSAGE DIGEST 5)

- designed by Ronald Rivest (the R in RSA)
- latest in a series of MD2, MD4
- produces a 128-bit hash value
- until recently was the most widely used hash algorithm
  - ❖ in recent times have both brute-force & cryptanalytic concerns
- specified as Internet standard RFC1321
- **MD5 Overview**

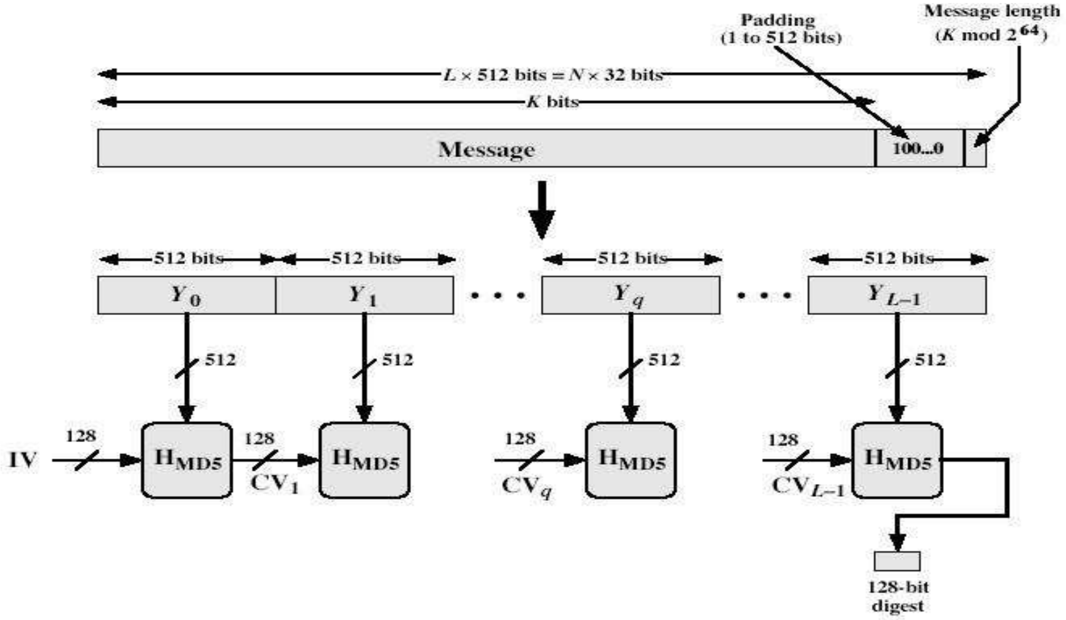
  1. pad message so its length is 448 mod 512
  2. append a 64-bit length value to message
  3. initialise 4-word (128-bit) MD buffer (A,B,C,D)
  4. process message in 16-word (512-bit) blocks:

     i. Compression function that consists of 4 rounds of processing labelled as HMD5

     ii. using 4 rounds of 16 bit operations on message block & buffer

     iii. Each round takes as i/p the current 512 bit block being processed(Yq) and 128bit buffer value ABCD and updates the content of the buffer

5. Output of the fourth round is added to the i/p to the first round(CVq) to produce CVq+1 Output: After all L 512bit blocks have been processed ,the output from the Lth stage is the 128bit msg digest



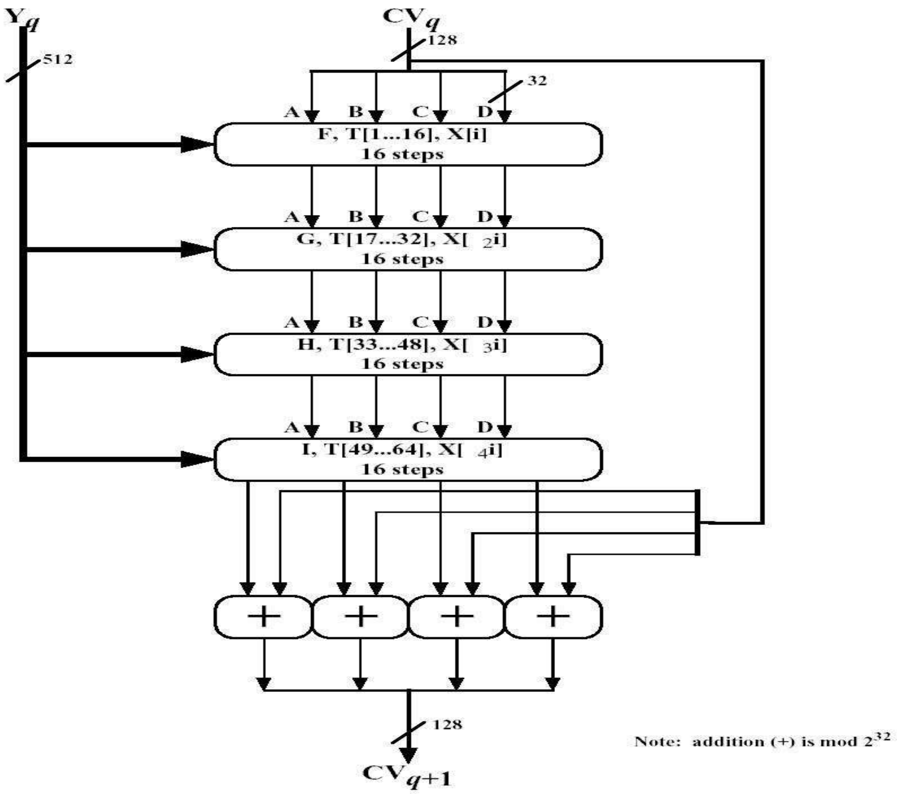


Figure 9.2 MD5 Processing of a Single 512-bit Block (MD5 Compression Function)

- MD5 Compression Function

❖ each round has 16 steps of the form:

a = b+((a+g(b,c,d)+X[k]+T[i])<<<s)

- A,b,c,d - four words of the buffer, in a specified order
- g-one of the primitive function F,G,H ,I
- <<<s - Circular shift of 32bit argument by s bit
- T[i] - ith 32 bit in matrix T
- +-addition modulo 2^{32}

❖ Each round mixes the buffer input with the next "word" of the message in a complex, non-linear manner. A different non-linear function is used in each of the 4 rounds (but the same function for all 16 steps in a round). The 4 buffer words (a,b,c,d) are rotated from step to step so all are used and updated. g is one of the primitive functions F,G,H,I for the 4 rounds respectively. X[k] is the kth 32-bit word in the current message block. T[i] is the ith entry in the matrix of constants T. The addition of varying constants T and the use of different shifts helps ensure it is extremely difficult to compute collisions.

❖ Four 32 bit registers

- A = 67452301
- B = EFCDAB89
- C = 98BADCFE
- D = 10325476

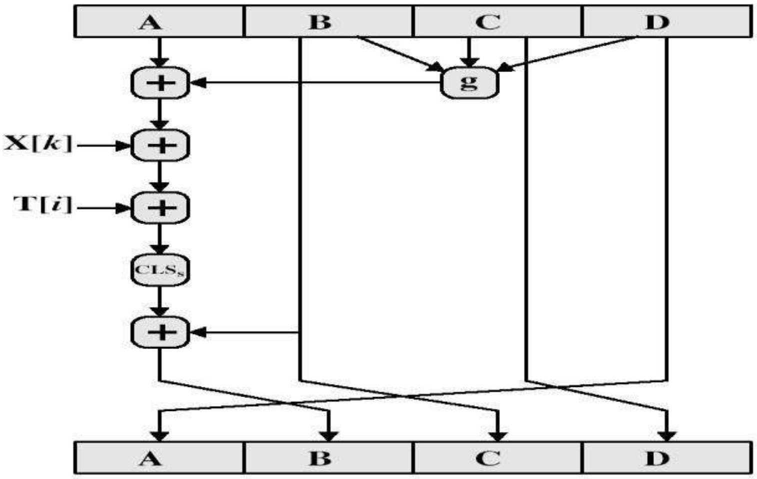
$$F(B,C,D) = (B \wedge C) \vee (\neg B \wedge D)$$

$$G(B,C,D) = (B \wedge D) \vee (C \wedge \neg D)$$

$$H(B,C,D) = B \oplus C \oplus D$$

$$I(B,C,D) = C \oplus (B \vee \neg D)$$

$\oplus, \wedge, \vee, \neg$ denote the XOR, AND, OR and NOT operations respectively.



- **Strength of MD5**
  - ❖ Every bit of the hash code is a function of every bit in the i/p
  - ❖ Rivest claims security is good
  - ❖ known attacks are:
    - – Berson , using differential cryptanalysis, possible in reasonable time to find two msg's that produce the same digest for single round. Bt he was not able to generalize attack to full round
    - – Boer & Bosselaers showed execution of MD5 on a single block of 512bits yeild the same output for two different input values in buffer ABCD->pseudocollision.

      Bt he was not able to extend this approach to a successful attack on MD5
    - – Dobbertin, operation of MD5 on a single 512bit block of i/p by finding another block that produces the same 128bit.He was not able to generalize it
  - ❖ conclusion is that MD5 looks vulnerable soon

- <u>SECURE HASH ALGORITHM (SHA-1)</u>

- SHA was designed by NIST & NSA in 1993, revised 1995 as SHA-1
- US standard for use with DSA signature scheme
  - ❖ standard is FIPS 180-1 1995, also Internet RFC3174
  - ❖ nb. the algorithm is SHA, the standard is SHS
- produces 160-bit hash values

- now the generally preferred hash algorithm
- based on design of MD4 with key differences
- **SHA Overview**
  1. pad message so its length is 448 mod 512
  2. append a 64-bit length value to message
  3. initialise 5-word (160-bit) buffer (A,B,C,D,E) to

     A= 67452301

     B= EFCDAB89

     C= 98BADCFE

     D= 10325476

     E= C3D2E1F0

  4. process message in 16-word (512-bit) chunks:
     a. Consists of four rounds of processing of 20 steps each
     b. Four rounds uses different primitive locgical func,f1,f2,f3,f4
     c. Each round takes as i/p the current 512bit block being processed(Yq) and 160bit buffer value A,B,C,D,E and updates buffer
  5. Each round uses an additive constant Kt where 0<=t<=79,indicates one of the 80 steps across five rounds
  6. Output of the 4th round is added to the i/p of the first round(CVq) to produce Cq+1
  7. **Output**: After L512 bit blocks have been processed, the output from Lth stage is 160bit msg digest

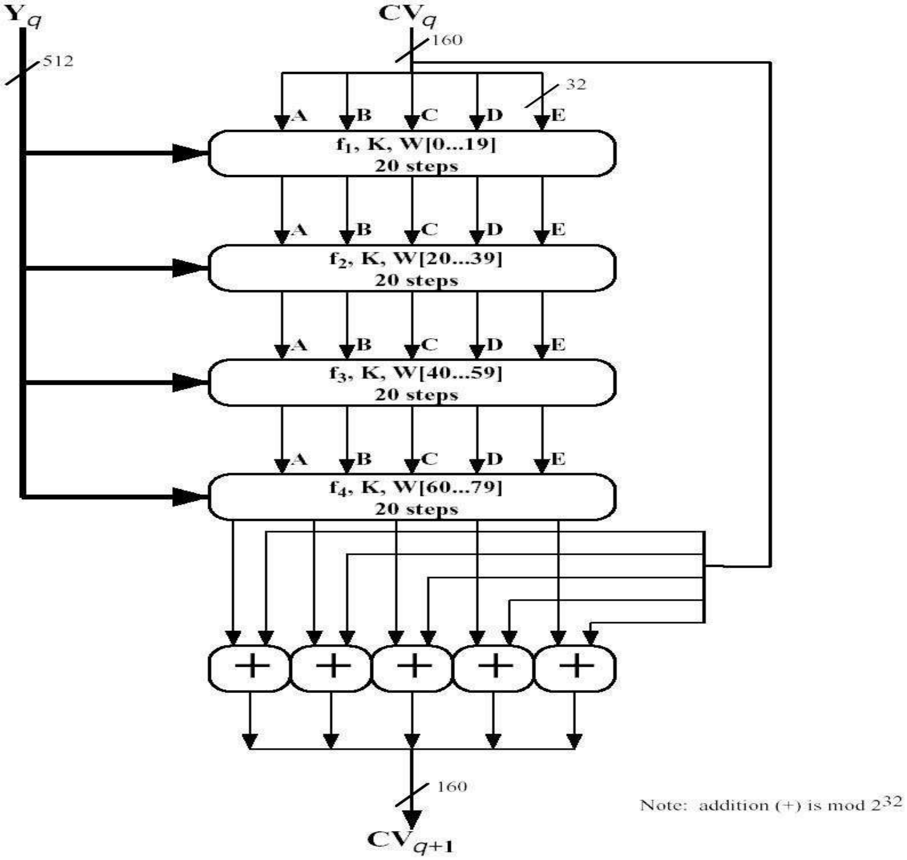


Figure 9.5 SHA-1 Processing of a Single 512-bit Block (SHA-1 Compression Function)

- **SHA-1 Compression Function**
  - ❖ Each round has 20 steps which replaces the 5 buffer words thus:

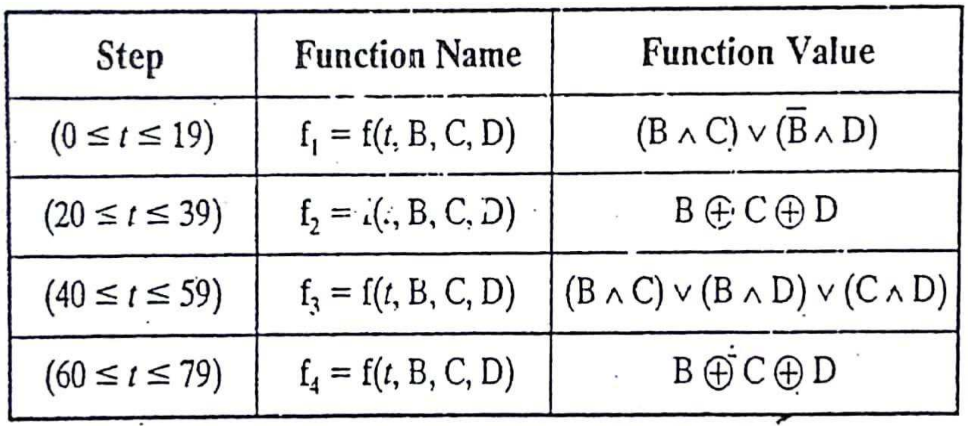
    (A,B,C,D,E) <-(E+f(t,B,C,D)+(A<<5)+W_t+K_t),A,(B<<30),C,D)

    - a,b,c,d refer to the 4 words of the buffer
    - t is the step number
    - f(t,B,C,D) is nonlinear function for round
    - W_t is derived from the message block
    - K_t is a constant value derived from sin
  - ❖ **Distinct constant values for each round**

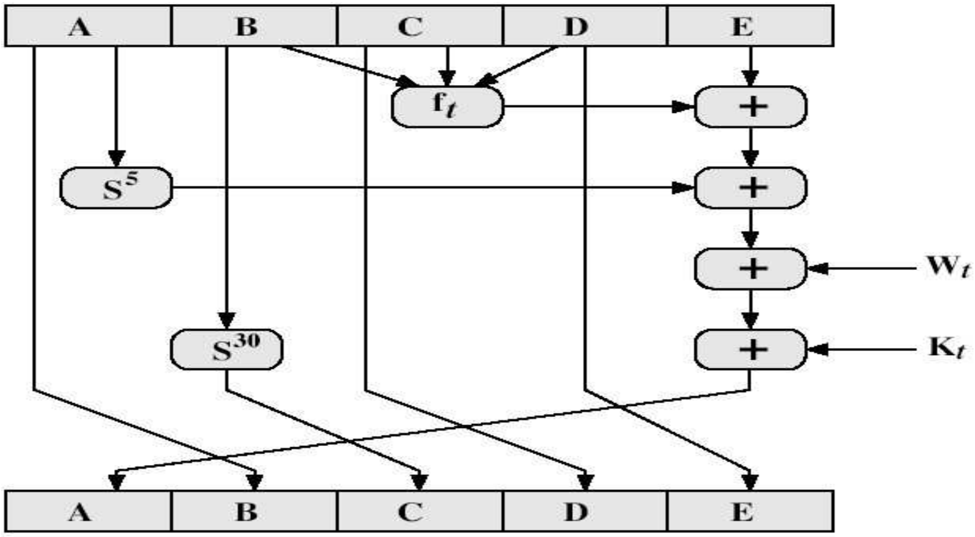
Hexadecimal	Step number
Kt = 5A827999	0<=t<=19

Kt = 6ED9EBA1	20<=t<=39
Kt = 8F1BBCDC	40<=t<=59
Kt = CA62C1D6	60<=t<=79

- **Functions for each round**



Step	Function Name	Function Value
$(0 \leq t \leq 19)$	$f_1 = f(t, B, C, D)$	$(B \wedge C) \vee (\bar{B} \wedge D)$
$(20 \leq t \leq 39)$	$f_2 = f(t, B, C, D)$	$B \oplus C \oplus D$
$(40 \leq t \leq 59)$	$f_3 = f(t, B, C, D)$	$(B \wedge C) \vee (B \wedge D) \vee (C \wedge D)$
$(60 \leq t \leq 79)$	$f_4 = f(t, B, C, D)$	$B \oplus C \oplus D$



- **SHA-1 verses MD5**
  - brute force attack is harder (160 vs 128 bits for MD5)
  - not vulnerable to any known attacks (compared to MD4/5)
  - a little slower than MD5 (80 vs 64 steps)
  - both designed as simple and compact
  - optimised for big endian CPU's (vs MD5 which is optimised for little endian CPU’s)

SECURITY OF HASH FUNCTIONS AND MACS.

with symmetric and public key encryption, attacks on hash functions and MACs into two categories:

- Brute force attacks.
- cryptanalysis.

I. BRUTE - FORCE ATTACKS.

(1) HASH FUNCTIONS

- The strength of a hash function against brute-force attack depends solely on the length of the hash code produced by the algorithm.
- Properties of hash functions:
  - (a) One-way: for any given code 'h' it is computational infeasible to find x- such that $H(x) = h$.
  - (b) weak collision resistance: for any given block x it is computationally infeasible to find $y \neq x$ with $H(y) = H($
  - (c) strong collision resistance: It is computationally infeasible to find any pair (x, y) such that $H(x) = H(y)$.

For a code of length 'n', the level of effort required is proportional to the following:

One way - 2^n

weak collision resistance - 2^n

strong collision resistance - $2^{n/2}$.

- If strong collision resistance is required, then the value $2^{n/2}$ determines the strength of the hash code against brute-force attack.

(ii) MESSAGE AUTHENTICATION CODES.

- A brute-Force attack on a MAC is more difficult because it requires known message MAC pairs.
- If an attacker can determine the MAC key, then it is possible to generate a valid MAC value for any input 'x'.
- Suppose the key size is 'k' bits and that the attacker has one known text-MAC pair

Then the attacker can compute the n-bit MAC on the known text for all possible keys.

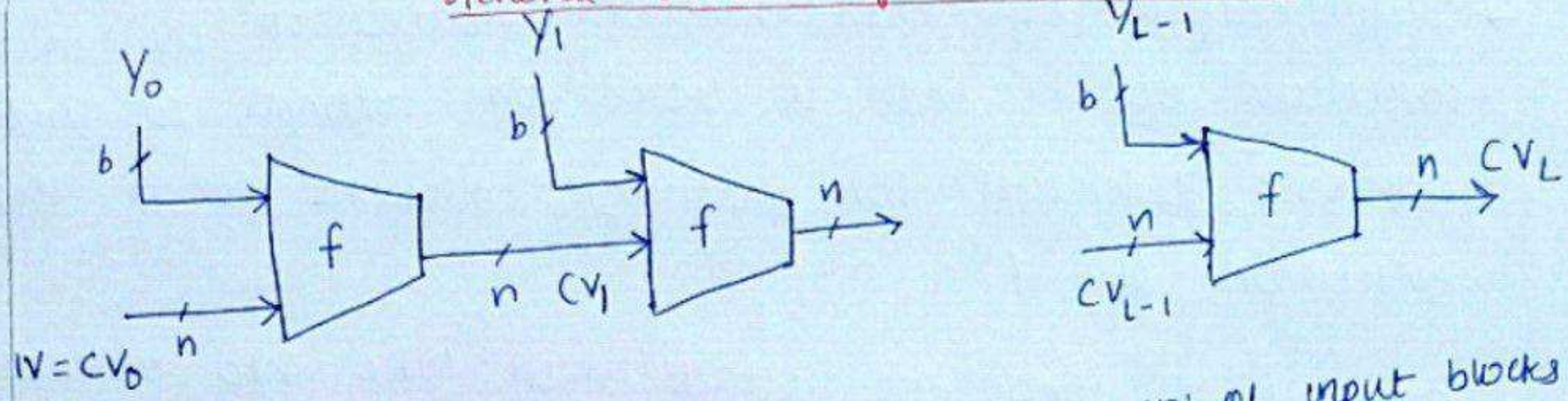
- At least one key is guaranteed to produce the known text-MAC pair. This phase of the attack takes a level of effort proportional to 2^k.
- An attacker can also work on the MAC value without attempting to recover the key. The objective is to generate a valid MAC value for a given message or to find a message that matches a given MAC value. In either case, the level of effort is comparable to that for attacking the one way or weak collision resistant property of a hash code or 2^n.

II CRYPTANALYSIS.

- An ideal hash or MAC algorithm will require a cryptanalytic effort greater than or equal to the brute-force effort.

(i) Hash functions

General structure of secure Hash code



IV → initial value

CV → chaining variable

Y_i → i^{th} input block.

f → compression algorithm

L → no: of input blocks

n → length of hash code

b → lengths of input block.

→ The hash function takes an input message and partitions into 'L' fixed sized blocks of 'b' bits each.

→ The final block is padded to 'b' bits (if necessary)

→ The final block also includes the value of the total length of the input to the hash function.

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

→ The hash algorithm involves a repeated use of a compression function 'f' that takes two inputs [an n-bit input from the previous step, called the chaining variable, and a 'b' bit block] and produces an n-bit output.

→ The final value of the chaining variable is the hash value

→ The hash function can be written as:

$$CV_0 = IV = \text{initial } n\text{-bit value}$$

$$CV_i = f(CV_{i-1}, Y_{i-1}) \quad ; \quad 1 \leq i \leq L$$

H [redacted] ' consists of blocks $Y_0, Y_1 \cdots Y_L$

where input to the hash funct is a msg M

→ cryptanalysis of hash functions focuses on the internal structure of 'f' and is based on attempts to find efficient techniques for producing collisions for a single execution of 'f'.

→ once it is done, the attack must take into account the fixed value of IV.

→ The attack on 'f' depends on exploiting its internal structure

→ [Note: for any hash function there must exist collisions, because we are mapping a message of length at least equal to the block size 'b' into a hash code of length 'n', where $b > n$.

(ii) MESSAGE AUTHENTICATION CODES

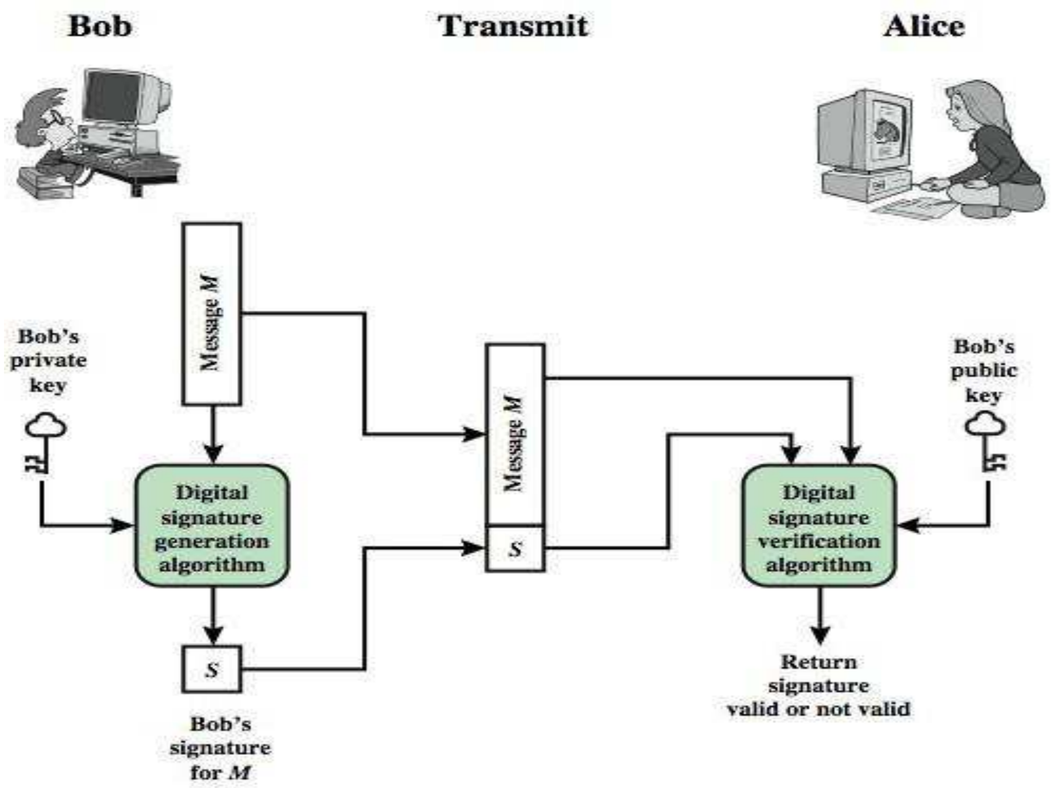
→ As there is more variety in the structure of MACs than in hash functions, so it is difficult to generalize about the cryptanalysis of MACs.

- **DIGITAL SIGNATURE**

  - Message authentication protects two parties who exchange messages from any third party. However, it does not protect the two parties against each other.
  - A digital signature is analogous to the handwritten signature, and provides a set of security capabilities that would be difficult to implement in any other way.
  - Digital Signature is an authentication mechanism that enables the creator of a msg to attach a code that acts as a signature.
  - The signature is formed by taking the hash of the msg and encrypting the msg with creators private key.
  - Signature guarantees the source and integrity of the msg
  - **PROPERTIES:**
    - It must verify the author and the date and time of the signature
    - It must to authenticate the contents at the time of the signature
    - It must be verifiable by third parties, to resolve disputes
  - **REQUIREMENTS FOR A DIGITAL SIGNATURE**
    - Must be a bit pattern that depends on the msg being signed
    - Must use some information unique to the sender, to prevent both forgery and denial
    - Must be relatively easy to produce digital signature
    - Must be relatively easy to recognize and verify digital signature
    - Must be computationally infeasible to forge a digital signature
    - Must be practical to retain a copy of the digital signature in storage
  - Two approaches has been proposed for digital signature:
    - Direct Digital Signature
    - Arbitrated Digital Signature
  - **DIRECT DIGITAL SIGNATURE**
    - Involves only the communication parties: Source and Destination
    - Destination knows the public key of the source
    - Formed by encrypting the entire msg with senders private key or by encrypting a hash code of the msg with senders private key
    - Confidentiality can be obtained by further encrypting the entire msg plus signature with either receivers public key or shared secret key
    - important that sign first then encrypt message & signature
    - **Drawback:** Security depends on the senders private key
      - If the sender wishes to deny sending a particular msg ,sender can claim the private key was stolen or lost or someone else forged her signature
      - To avoid this, require every signed msg to include a timestamp

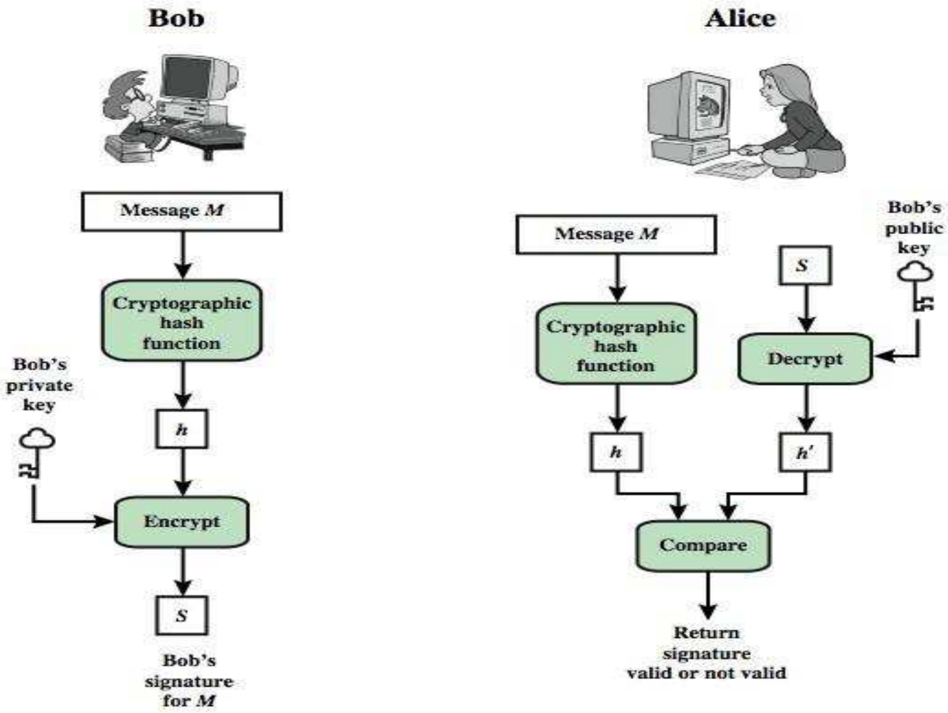
- Another threat, private key might actually stolen from X at time ‘T’

- Opponent can then send a msg signed with X’s signature and time stamped with a time before or equal to ‘T’
- **Digital Signature Model**


Bob
Transmit
Alice
Bob’s private key
Message *M*
Digital signature generation algorithm
S
Bob’s signature for *M*
Message *M*
S
Bob’s public key
Digital signature verification algorithm
Return signature valid or not valid

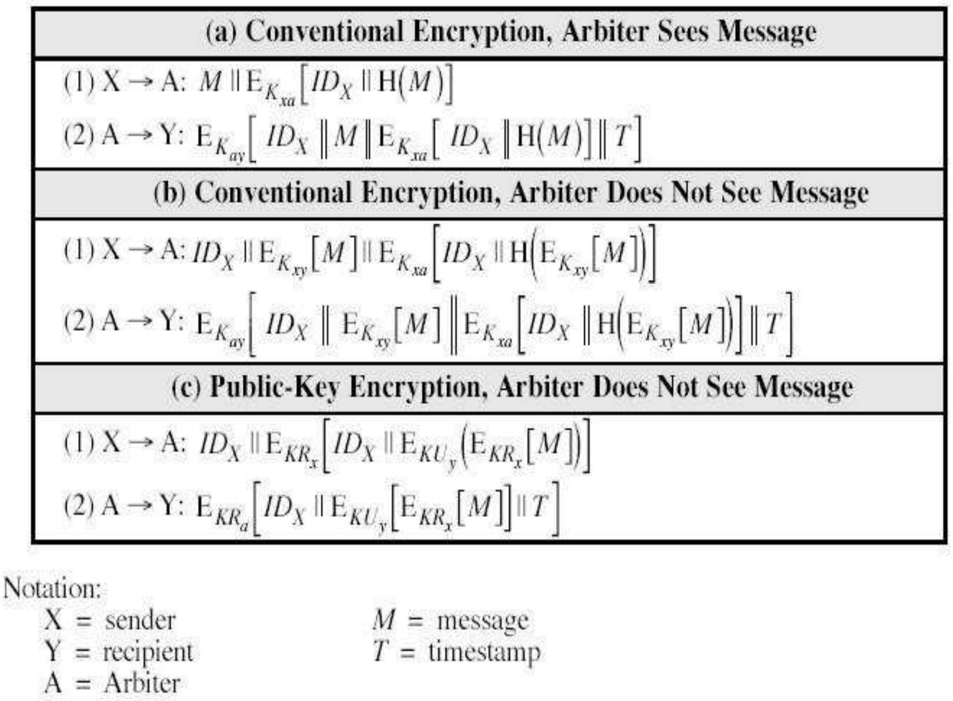
- Bob can sign a message using a digital signature generation algorithm. The inputs to the algorithm are the message and Bob's private key. Any other user, say Alice, can verify the signature using a verification algorithm, whose inputs are the message, the signature, and Bob's public key.

- **Digital Signature Model**



Arbitrated

- Every signed msg from a sender X to a receiver Y goes first to an arbiter A, who subjects the msg and its signature to a number of tests to check its origin and contents
- Msg is then dated and sent to Y with an indication that it has been verified to the satisfaction of the arbiter



(a) Conventional Encryption, Arbiter Sees Message
(1) X → A: $M \| E_{K_{xa}}[ID_X \| H(M)]$
(2) A → Y: $E_{K_{ay}}[ID_X \| M \| E_{K_{xa}}[ID_X \| H(M)] \| T]$
(b) Conventional Encryption, Arbiter Does Not See Message
(1) X → A: $ID_X \| E_{K_{xy}}[M] \| E_{K_{xa}}[ID_X \| H(E_{K_{xy}}[M])]$
(2) A → Y: $E_{K_{ay}}[ID_X \| E_{K_{xy}}[M] \| E_{K_{xa}}[ID_X \| H(E_{K_{xy}}[M])] \| T]$
(c) Public-Key Encryption, Arbiter Does Not See Message
(1) X → A: $ID_X \| E_{KR_x}[ID_X \| E_{KU_y}(E_{KR_x}[M])]$
(2) A → Y: $E_{KR_a}[ID_X \| E_{KU_y}[E_{KR_x}[M]] \| T]$

Notation:

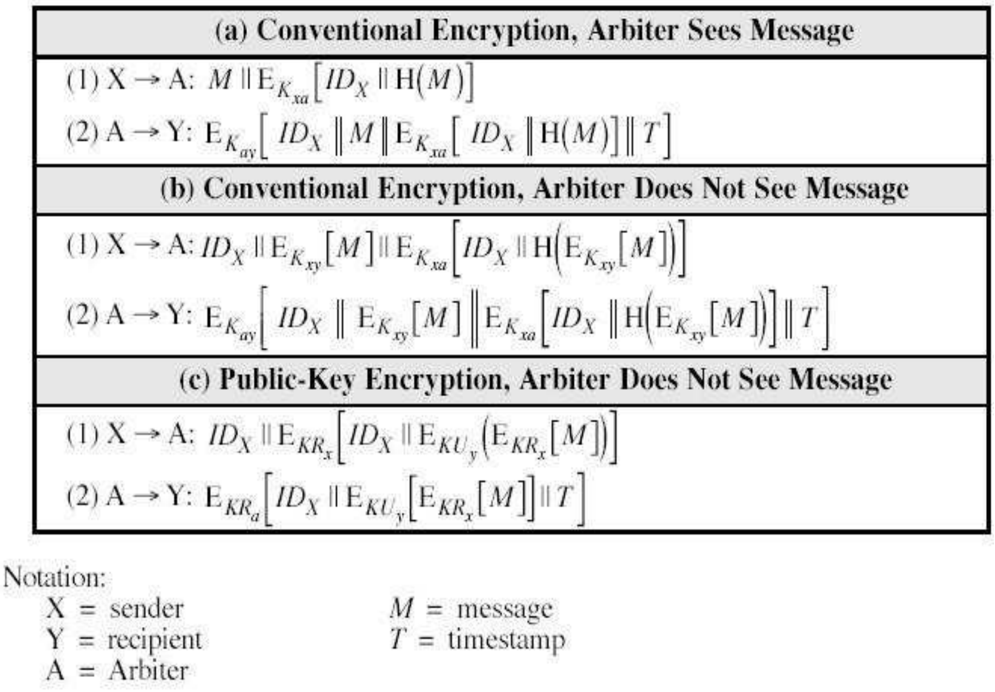
X = sender	M = message
Y = recipient	T = timestamp
A = Arbiter	

a)

- Sender X and arbiter A share a secret key K_{xa}
- A and Y share secret key K_{ay}
- X constructs a msg M and computes its hash value H(M)
- X transmits msg plus a signature to A
- Signature consists of identifier of X, IDx plus hash value, all encrypted using K_{xa}
- A decrypts the signature and checks the hash value to validate the msg
- A transmits the msg to Y encrypted with K_{ay}
- Msg includes IDx, the original msg from X, the signature and a timestamp
- Y can decrypt this to recover the msg and signature
- Timestamp informs Y that this msg is timely and not replay
- In case of dispute Y, who received msg M from X sends following msg to A:

$$E(K_{ay}[IDx||M||E(K_{xa}[IDx||H(M)])])$$

- Arbiter uses K_{ay} to recover IDx, M and signature, then uses Kxa to decrypt the signature and verify hash code
- Both sides have high degree of trust in A
  - X must trust A not to reveal K_{xa} and not to generate false signature of the form $E(K_{xa}[IDx||H(M)])$
  - Y trust A to send E(Kay[IDx||M||E(Kxa[Idx||H(M)]||T])only if the hash value is correct and the signature was generated by X
  - Both sides must trust A to resolve dispute fairly



(a) Conventional Encryption, Arbiter Sees Message
(1) X → A: $M \| E_{K_{xa}}[ID_X \| H(M)]$
(2) A → Y: $E_{K_{ay}}[ID_X \| M \| E_{K_{xa}}[ID_X \| H(M)] \| T]$
(b) Conventional Encryption, Arbiter Does Not See Message
(1) X → A: $ID_X \| E_{K_{xy}}[M] \| E_{K_{xa}}[ID_X \| H(E_{K_{xy}}[M])]$
(2) A → Y: $E_{K_{ay}}[ID_X \| E_{K_{xy}}[M] \| E_{K_{xa}}[ID_X \| H(E_{K_{xy}}[M])] \| T]$
(c) Public-Key Encryption, Arbiter Does Not See Message
(1) X → A: $ID_X \| E_{KR_x}[ID_X \| E_{KU_y}(E_{KR_x}[M])]$
(2) A → Y: $E_{KR_a}[ID_X \| E_{KU_y}[E_{KR_x}[M]] \| T]$

Notation:

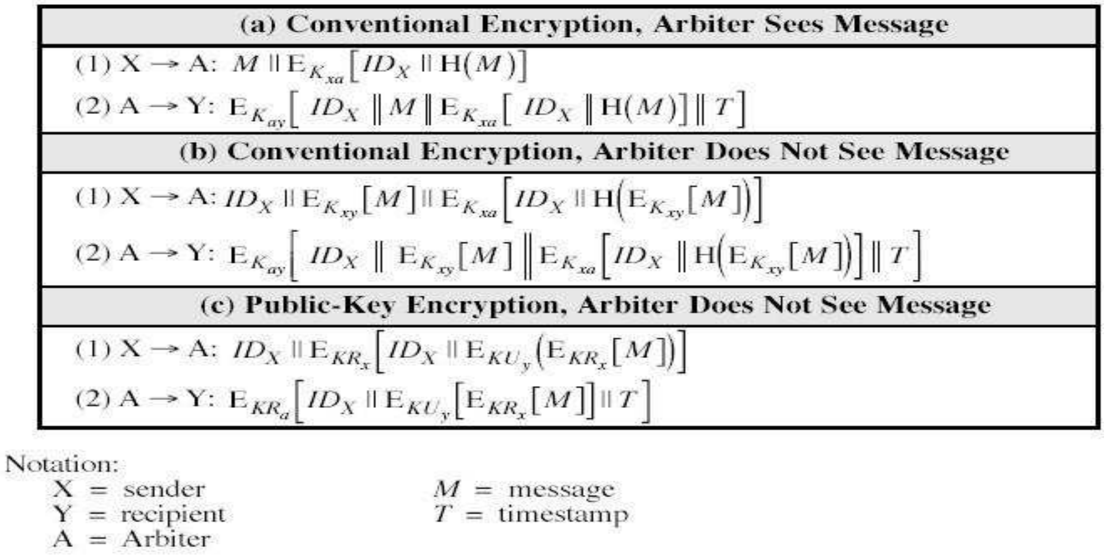
X = sender	M = message
Y = recipient	T = timestamp
A = Arbiter	

b)

- Provides confidentiality
- X and Y share the secret key Kxy
- X transmits an identifier , a copy of msg encrypted with Kxy and signature to A
- Signature consists of the identifier plus the hash value of the encrypted msg,all encrypted using Kxa
- A decrypts the signature and checks the hash value to validate the message

- A is working only with encrypted version of the msg and is prevented from reading it

- A then transmits everything that it received from X, plus a timestamp, all encrypted with Kay to Y



(a) Conventional Encryption, Arbiter Sees Message
(1) X → A: $M \parallel \mathrm{E}_{K_{xa}}\left[ID_X \parallel \mathrm{H}(M)\right]$
(2) A → Y: $\mathrm{E}_{K_{ay}}\left[ID_X \parallel M \parallel \mathrm{E}_{K_{xa}}\left[ID_X \parallel \mathrm{H}(M)\right] \parallel T\right]$
(b) Conventional Encryption, Arbiter Does Not See Message
(1) X → A: $ID_X \parallel \mathrm{E}_{K_{xy}}[M] \parallel \mathrm{E}_{K_{xa}}\left[ID_X \parallel \mathrm{H}\left(\mathrm{E}_{K_{xy}}[M]\right)\right]$
(2) A → Y: $\mathrm{E}_{K_{ay}}\left[ID_X \parallel \mathrm{E}_{K_{xy}}[M] \parallel \mathrm{E}_{K_{xa}}\left[ID_X \parallel \mathrm{H}\left(\mathrm{E}_{K_{xy}}[M]\right)\right] \parallel T\right]$
(c) Public-Key Encryption, Arbiter Does Not See Message
(1) X → A: $ID_X \parallel \mathrm{E}_{KR_x}\left[ID_X \parallel \mathrm{E}_{KU_y}\left(\mathrm{E}_{KR_x}[M]\right)\right]$
(2) A → Y: $\mathrm{E}_{KR_a}\left[ID_X \parallel \mathrm{E}_{KU_y}\left[\mathrm{E}_{KR_x}[M]\right] \parallel T\right]$

Notation:

X = sender
Y = recipient
A = Arbiter

M = message
T = timestamp

c)

- X double encrypts a msg M first with X's private key KRx and then with Y's public key KUy

- This signed msg,together with X's identifier is encrypted again with KRx and together with IDx is sent to A

- Double encrypted msg is secure from arbiter

- A can decrypt the outer encryption to assure that the msg must have come from X

- Then A transmits a msg to Y, encrypted with KRa

- The msg includes IDx, double encrypted msg and a timestamp

- **Advantages**:

1. No information is shared among parties before communication, preventing alliances to defraud

2. No incorrectly dated msg can be sent

3. Content of the msg from X to Y is secret from A and anyone else

- **Drawback**: involves encryption of the msg twice with public key algorithm

AUTHENTICATION PROTOCOLS

- Used to convince parties of each other's identity and to exchange session keys
- May be mutual authentication and one-way authentication
- **MUTUAL AUTHENTICATION:**
  - ❖ Enable communicating parties to satisfy themselves mutually about each other's identity and to exchange session keys
  - ❖ key issues are
    - confidentiality – to protect session keys
    - timeliness – to prevent replay attacks
  - ❖ **Replay Attacks**
    - **Simple Replay:** Opponent copies a msg and replays it later
    - **Repetition that can be logged:** Opponent replay a time stamped msg within the valid time windows
    - **Repetition that cannot be detected: Original** msg is suppressed and did not arrive its destination; only the replay msg arrives
    - **Backward replay without modification**: Replay back to the msg sender
  - ❖ In symmetric encryption sender cannot differentiate msg sent and msg received based on the content
  - ❖ Solution: Attach a sequence no: to each msg
  - ❖ **Difficulty:**Require each party to keep track of the last sequence no.
  - ❖ So sequence no is not used
  - ❖ Following two approaches are used:
    - **Timestamp**: Party A accepts a msg as fresh only if the msg contains a timestamp that, in A's judgment ,is close enough to A's knowledge of current time. Clocks should be synchronized
    - **Challenge/Response** : Party A, expecting a fresh msg from B,first sends B a nonce(challenge) and requires that the subsequent msg(response) received from B contain the correct nonce value
  - ❖ Timestamp approach not used-connection oriented because:
    1. Protocol is needed to maintain synchronization among processor clock

2. Successful attack will happen if there is temporary loss of synchronization
3. Distributed clock cannot be expected to maintain precise synchronization

- Challenge-Response is unsuitable for connectionless

❖ Requires the overhead of a handshake before any connectionless transmission

❖ **Symmetric Encryption Approaches**

- Involves the use of Trusted key distribution center
- Each party shares ‘master key’ with KDC
- KDC is responsible for generating keys to be used for a short time over a connection between the two parties
- Needham & Schroeder protocol:
- For secret key distribution using a KDC
- original third-party key distribution protocol
- for session between A B mediated by KDC
- protocol overview is:

1. A->KDC: $ID_A \,||\, ID_B \,||\, N_1$

2. KDC -> A: $E(K_a,[K_s||ID_B||N_1||\ E(K_b,[K_s||ID_A])])$

3. A -> B: $E(K_b, [K_s||ID_A])$

4. B -> A: $E(K_s, [N_2])$

5. A -> B: $E(K_s, [f(N_2)])$

- Each user shares a unique master key with KDC
- A wishes to establish logical connection with B and requires session key
- A has a master key Ka shared between A and KDC
- Similarly B and KDC, Kb

❖ **Steps**

- A issues a request to KDC for session key . Msg includes identity of A and B and a unique identifier N1 for this transaction

- KDC responds->msg encrypted with Ka,only A can read it. Msg also includes

  - One time session key Ks
  - Original request msg,to match this response with appropriate request

- Msg also includes 2 items for B:

  - One time session key Ks

  - Identifier of A,IDa

  - These two items encrypted with Ekb

  - They are to be sent to B to establish the connection and prove A's identity

  - A stores session key and forward to B the information that originated at the KDC for B

  - Session key has been securely delivered to A and B,and they begin their protected exchange.

  - B sends a nonce ,N2 to A

  - Using EKs,A responds with f(N2),f is a function that performs transformation on N2

- Still vulnerable to replay attack

- Modification to Needham & Schroeder protocol(Denning Protocol), includes timestamp to step 2 and 3

1.A->KDC: ID_A || ID_B

2. KDC -> A: E(K_a,[K_s||ID_B||T|| E(K_b,[K_s||ID_A||T])])

3. A -> B: E(K_b, [K_s||ID_A||T])

4. B -> A: E(K_s, [N_1])

5. A -> B: E(K_s, [f(N_1)])

- Increased security compared to previous
- Drawback-Requires reliance on clocks that are synchronized throughout the network
- Problem occurs when a senders clock is ahead of the intended recipients clock

  - Opponent can intercept a msg from the sender and replay it later when the timestamp in the msg become current at the recipients site , results in unexpected results->suppress replay attack

  - Solution:Parties regularly check their clock against the KDC's clock

    or,Rely on handshaking protocols using nonce

    - To overcome all, new protocol:

1. A->B $ID_A||Na$
2. B->KDC $ID_B||Nb||E(Kb[ID_A||Na||Tb])$
3. KDC->A $E(Ka[ID_B ||Na||Ks||Tb])||E(Kb[ID_A||Ks||Tb])||Nb$
4. A->B E(Kb[IDA||Ks||Tb])||E(Ks , Nb)

Steps:

1. A initiate by generating a nonce Na and sending that plus its identifier to B. Nonce will be returned to A in an encrypted msg that includes session key, assuring A of its timeliness

2. B alerts KDC that a session key is needed . Msg to KDC includes its identifier and a nonce, Nb. Nonce will be returned to B in an encrypted msg that includes the session key, assuring Bof its timeliness

3. KDC passes on to A B's nonce and block encrypted with the secret key that B shares with the KDC.

   - Block serves as a "ticket" that can be used by A for subsequent authentications

   - KDC also sends to A a block encrypted with secret key shared by A and KDC. Block verifies B has received A's initial msg's(IDB).It provides A with session key (Ks)and time limit on its use

4. A transmits the ticket to B,together with B's nonce,the latter encrypted with the session key.

- Ticket provides B with secret key that is used to decrypt E(Ks,Nb)to recover the nonce

❖ PUBLIC –KEY ENCRYPTION APPROACHES

- A protocol using timestamp is:

1. A->AS $ID_A||ID_B$
2. AS->A $E(KRas[ID_A||KUa||T])||E(KRas\ [ID_B||KU_b||T])$
3. A->B $E(KRas[ID_A||KUa||T])||E(KRas[ID_B||KU_b|T])||EKU_b[E(KRas[Ks||T])$

- Central system-Authentication server provides public key certificates
- Session key is chosen and encrypted by A
- **Drawback**: Requires Synchronization of clocks
- Another approach by Woo and Lam

1. A $\prec$ KDC: $ID_A \parallel ID_B$

2. KDC $\prec$ A: $E(PR_{auth}[ID_B \parallel PU_b]\ )$

3. A $\prec$ B: $E(_{PUb}[N_a \parallel ID_A])$

4. B $\prec$ KDC: $ID_B \parallel ID_A \parallel E(PU_{auth}[N_a])$

5. KDC $\prec$ B: $E_{(PRauth}[ID_A||PU_a] \parallel E_{PUb}[E_{PRauth}\ [N_a \| K_s \|\ ID_B]]$

6. B $\prec$ A: $E_{PUa}[E_{PRauth}\ [N_a \| K_s \|\ ID_B] \parallel N_b]$

7. A $\prec$ B: $E_{ks}[N_b]$

- Steps
  1. A informs KDC of its intention to establish a connection with B
  2. KDC returns to A a copy of B's public key certificate
  3. Using B's public key,A informs B of its desire to communicate and send a nonce Na
  4. B asks KDC for A's public key certificate and request a session key.B includes A's nonce that the KDC can stamp the session key with that nonce. Nonce is protected using KDC's public key
  5. KDC returns to B a copy of A's public key certificate plus{Na,Ks,ID_B}
  6. Above triple encrypted with KDC's private key is relayed to A,together with a nonce Nb generated by B

7. Assures B of A’s knowledge of session key

- **One-way Authentication**

  - ❖ Recipient wants some assurance that the msg is from the alleged sender

  - ❖ **Symmetric Encryption Approach**

1. A->KDC: $ID_A \,||\, ID_B \,||\, N_1$
2. KDC -> A: $E(K_a, [K_s||\mathit{ID_B}||N_1 \,||\, E(K_b,[K_s||\mathit{ID_A}])])$
3. A -> B: $E(K_b, [K_s||\mathit{ID_A}]) \,||\, E(K_s, M)$

   - does not protect from replay attack

   - Guarantees that only the intended recipient of a msg will be able to read it

  - ❖ **Public key encryption Approaches:**

    - Confidentiality prime concern

      - $A < B$: $E_{KUb}[Ks] \;||\; E_{Ks}[M]$
        - o Msg is encrypted with a one-time secret key
        - o Encrypting one-time key with B’s public key.Only B can decrypt it

    - Authentication prime concern

      - A->B: $M \,||\, E_{KRa}[H(M)]$

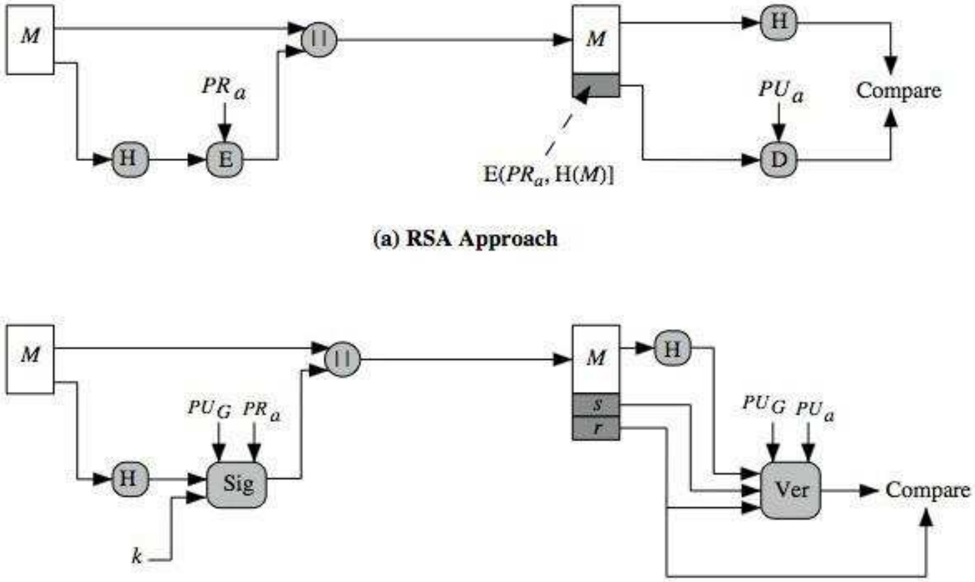
    - To be more secure both msg and signature are encrypted with recipients public key
      - A->B: $E_{KUb}\,[M \,||\, E_{KRa}[H(M)]]$
    - To make it more timely
      - A->B: $M \,||\, E_{KRa}[H(M)] \,||\, E_{KRas}[T||ID_A||KU_a]$

DIGITAL SIGNATURE STANDARD

- o uses the SHA hash algorithm

- o designed by NIST & NSA in early 90's

- DSS is the standard, DSA is the algorithm
- a variant on ElGamal and Schnorr schemes
- creates a 320 bit signature, but with 512-1024 bit security
- security depends on difficulty of computing discrete logarithms



(a) RSA Approach

(b) DSS Approach

- DSS use a hash function
- Hash code is provided as i/p to a signature fun along with a random number k
- Signature function also depends on the senders private key(PRa) and a global public key PUg
- Resulting signature consists of 2 parts : s and r
- At the receiving end, hash code of the incoming msg is generated
- This plus the signature is put input to a verification function
- Verification function also depends on global public key and senders public key
- Output of the verification function is a value that is equal to the signature component ‘r’ ,if signature is valid
- **DIGITAL SIGNATURE ALGORITHM (DSA)**
    - creates a 320 bit signature

- ❖ with 512-1024 bit security
- ❖ smaller and faster than RSA
- ❖ a digital signature scheme only
- ❖ security depends on difficulty of computing discrete logarithms
- ❖ **DSA Key Generation**
  - ▪ have shared global public key values (p,q,g):
    - ✓ choose 160-bit prime number q
    - ✓ choose a large prime p with $2^{L-1} < p < 2^L$
      - – where L= 512 to 1024 bits and is a multiple of 64
      - – such that q is a 160 bit prime divisor of (p-1)
    - ✓ choose $g = h^{(p-1)/q}$
      - – where 1<h<p-1 and $h^{(p-1)/q}$ mod p > 1
  - ▪ users choose private & compute public key:
    - ✓ choose random private key: x<q
    - ✓ compute public key: $y = g^x \bmod p$
- ❖ **DSA Signature Creation**
  - ▪ to **sign** a message M the sender:
    - ✓ generates a random signature key k, k<q
    - ✓ nb. k must be random, be destroyed after use, and never be reused
  - ▪ then computes signature pair:
    - ✓ $r = (g^k \bmod p) \bmod q$
    - ✓ $s = [k^{-1}(H(M)+ xr)] \bmod q$
  - ▪ sends signature (r,s) with message M
- ❖ **DSA Signature Verification**
  - ▪ having received M & signature (r,s)

- to **verify** a signature, recipient computes:

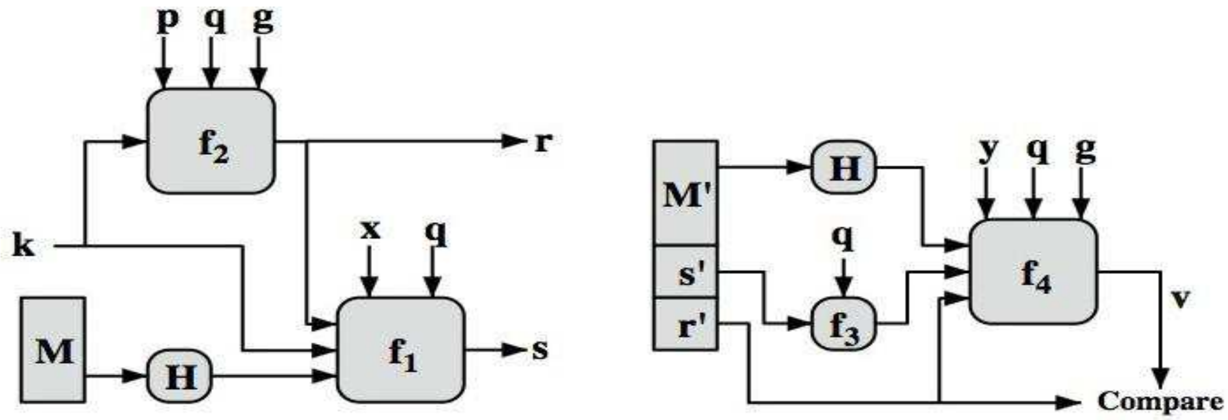
  $w = s^{-1} \bmod q$

  $u1= [H(M)w\ ]\bmod q$

  $u2= (rw)\bmod q$

  $v = [(g^{u1}\ y^{u2})\bmod\ p\ ]\bmod q$

- if v=r then signature is verified



$s = f_1(H(M), k, x, r, q) = (k^{-1}\ (H(M) + xr)) \bmod q$

$r = f_2(k, p, q, g) = (g^k \bmod p) \bmod q$

(a) Signing

$w = f_3(s', q) = (s')^{-1} \bmod q$

$v = f_4(y, q, g, H(M'), w, r')$

$= ((g^{(H(M')w) \bmod q}\ y^{r'w \bmod q}) \bmod p) \bmod q$

(b) Verifying

V^th module

Electronic Mail Security

Two schemes used for email security:-

↳ PGP (Pretty Good Privacy)

↳ S/MIME (Secure/Multipurpose Internel Mail Extension)

PGP

PGP is mainly used for personal email security. It is an effort of a single person Phil Zimmermann PGP provides confidentiality and authentication service that can be used for email and file storage application.

Features:-

1). Uses best cryptographic mechanisms, includes RSA, DSS and DH for public key encryption. CAST-128, IDGA and 3DES for symmetric encryption. SHA-1 for hash coding.

2). Available free worldwide via the Internet.

3) Platform intependent

4) low-cost

5) It was not developed by , nor is it controlled by government or standard organization.

6). PGP is now an Internet standard RFC3156.

Operational Description

5 Services of PGP :-

1. Authentication
2. Confidentiality
3. Compression
4. Email compatibility
5. Segmentation.

Authentication

The digital signature service provided by PGP is :-

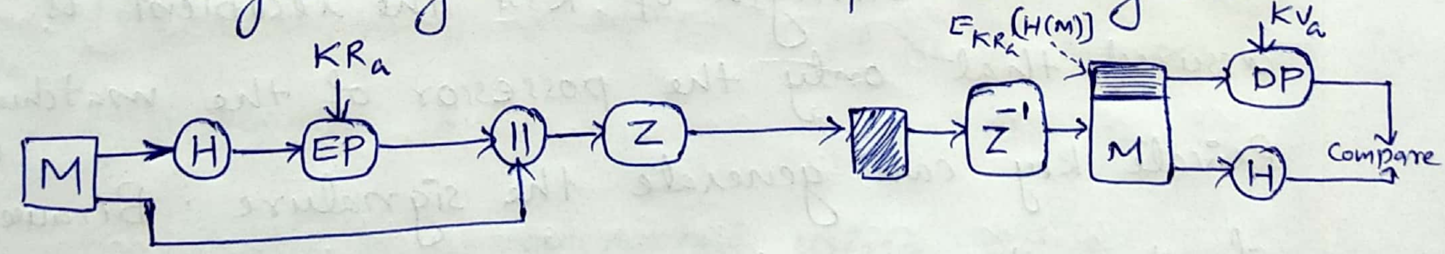


fig:- PGP authentication only.

where-

M - Plaintext Message

H - Hash function

EP - Public key encryption

DP - Publickey decryption

KR_a - Private key of user A

KU_a - publickey of user A

|| - Concatination

Z - compression using ZIP algo

Z^{-1} - Inverse compression.

Sequence is

1. Sender creates a message
2. SHA-1 is used to generate a 160-bit hash code of the message.
3. The hash code is encrypted with RSA using the sender's private key, and the result is prepended to the message.
4. The receiver uses RSA with sender's public key to decrypt and recover the hash code.
5. The receiver generates a new hash code for the message and compares it with the decrypted hash code. If the two match the message is accepted as authentic.

Here the combination of SHA-1 and RSA provides an effective digital signature scheme.
Due to the strength of RSA the recipient is assured that only the possessor of the matching private key can generate the signature. Because of the strength of SHA-1 the recipient is assured that no one else could generate a new message that matches the hash code and hence the signature of the original message.

2. Confidentiality

In PGP Confidtialy is provided by encrypting message to be transmitted or to be stored locally as files.

Confidentiality service provided by PGP :-

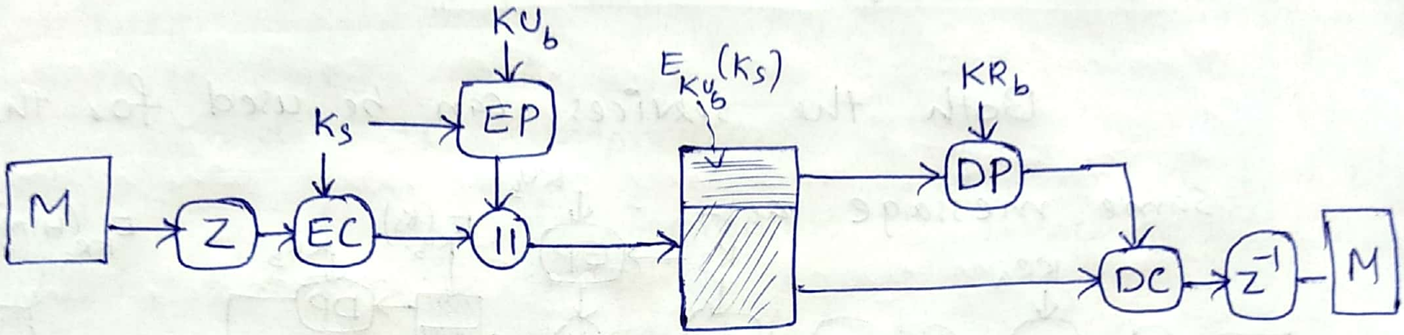


fig: PGP Confidentiality only.

where :- EC - Symmetric Encryption

DC - Symmetric Decryption

Ks - Session key used in Symmetric Encryption algo

<u>The sequence is :-</u>

1. The sender generates a message and a random 128-bit number to be used as a session key for this message only
2. The message is encrypted using CAST-128 (or IDEA or 3DES) with the session key
3. The session key is encrypted with RSA, using the recipient's public key, and is preponded to the message.

4. The receiver uses RSA with its private key to decrypt and recover the session key

5. The session key is used to decrypt the message.

Confidentiality and Authentication

Both the services can be used for the same message as:-

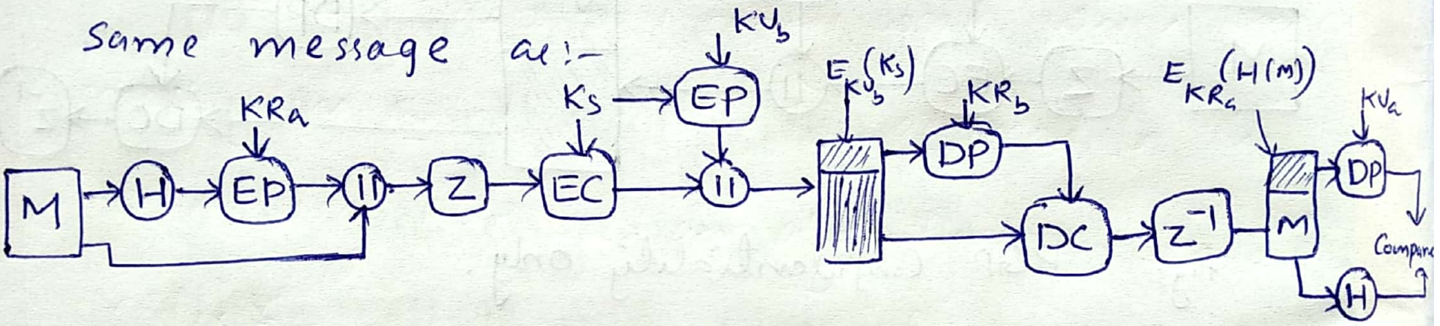


fig:- Confidentiality & authentication in PGP.

Here first a signature is generated for the Plaintext message and prepended to the message Then the plaintext message plus signature is encrypted using CAST-128 (or IDEA or 3DES) and the session key is encrypted using RSA.

ie when both services are used, the sender first sign the message with its own private key, then encrypts the message with a session key and then encrypts the session key with the recipient's public key.

3. Compression

PGP compresses the message after applying signature but before encryption. This has the benifits of saving space both for e-mail transmission and for file storage. Compression algo used in PGP is ZIP.

Z - indicates compression } placement of Z at Z^{-1} in algo is critical:-

Z^{-1} - indicates decompression

*① The signature is generated before compression for two reasons:-

a) It is preferable to sign an uncompressed message so it is free of the need for a compression algorithm for later verfication.

b) Different version of PGP produce different compressed forms. Applying the hash function and signature after compression would constrains all PGP implementations to the same version of the compressed algorithm.

② Message encryption is applied after compression to strengthen cryptographic security. Because the compressed message has less redundancy than the original plaintext, cryptanalysis is difficult.

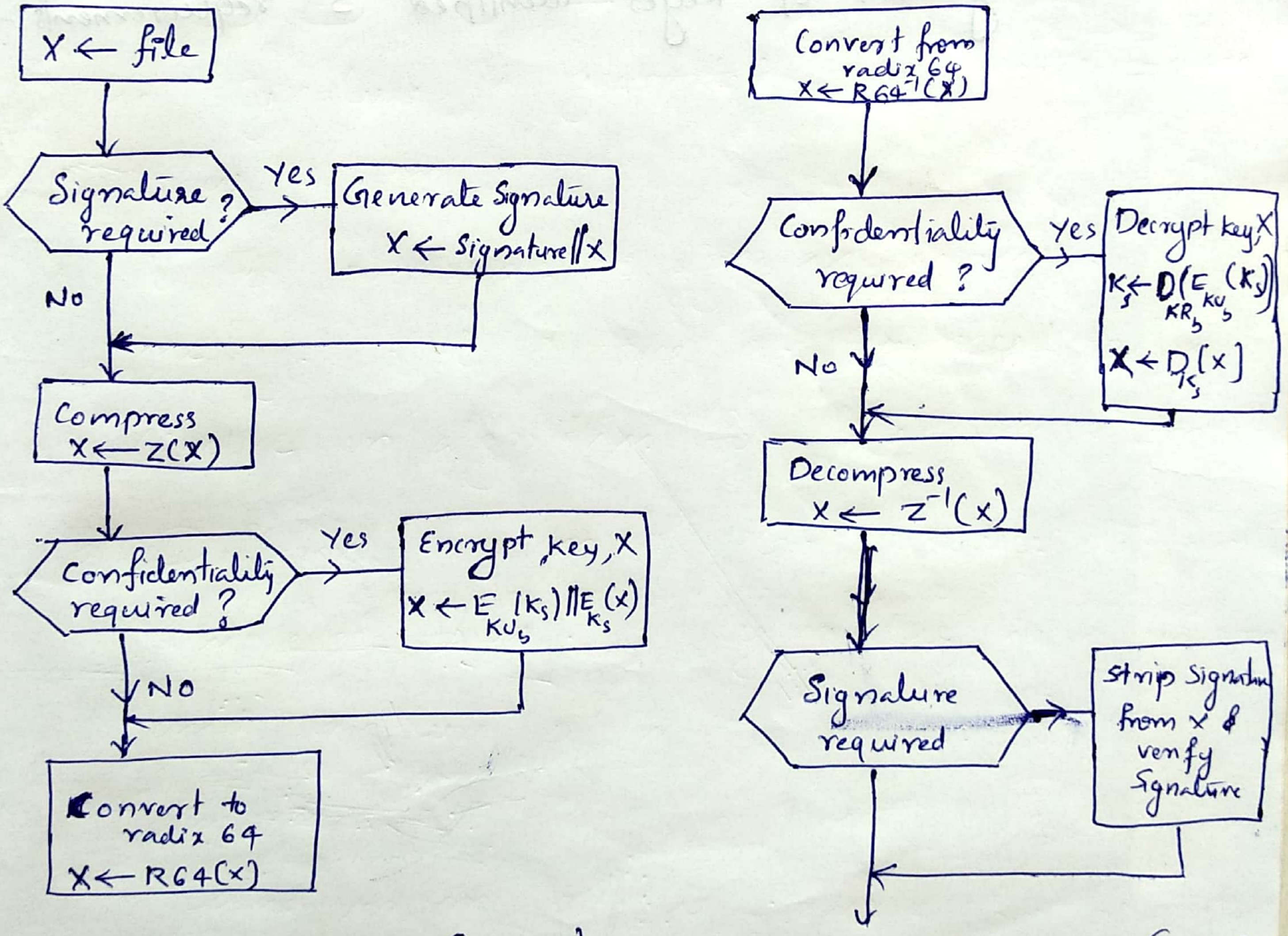
4. Email Compatibility

Many electronic mail systems only permit the use of blocks consisting of ASCII text. when PGP is used, at least part of the block to be transmitted is encrypted. This basically produce a sequence of arbitrary binary words which some mailsystem won't accept. To accomodate this restriction PGP uses an algorithm knoown as radix 64 which maps 6 bits of binary data into 8 bit ASCII character. Unfortunately this expands the message by 33% however with the compression algorithm overall compression will be about one third.

5. Segmentation

Email facilities are often restricted to a maximum message length. for excample, many of the facilities accessible throughout the internet impose a maximal length of 50,000 octets. Any msg longer than that must be broken up into smaller segments, each of which is mailed seperately.

To accomodate this restriction, PGP automatically subdivides a message that is too large into segment that are small enough to send via e-mail. The Segmentation is done after all of the other Processing including the radix-64 conversion. which is illustrated as follows:



(a) Transmission diagram (from A)

b) Reception diagram (to B)

fig:- Transmission & Reception of PGP messages.

→ IP Security

IP level security encompasses 3 functional areas:-

→ Authentication

→ confidentiality

→ Key Management.

(1) IP Security Overview

In IP-level security authentication and encryption are the necessary security features which has been issued as IPV6.

Applications of IPSec

IPSec provides the capability to secure communications across a LAN, across private and public WANs, and across the Internet.

Eg.- → Secure branch office connectivity over the internet.

→ Secure remote access over the Internet.

→ Establishing extranet and intranet connectivity with partners.

→ Enhancing electronic commerce security.

Benifits of IPSec

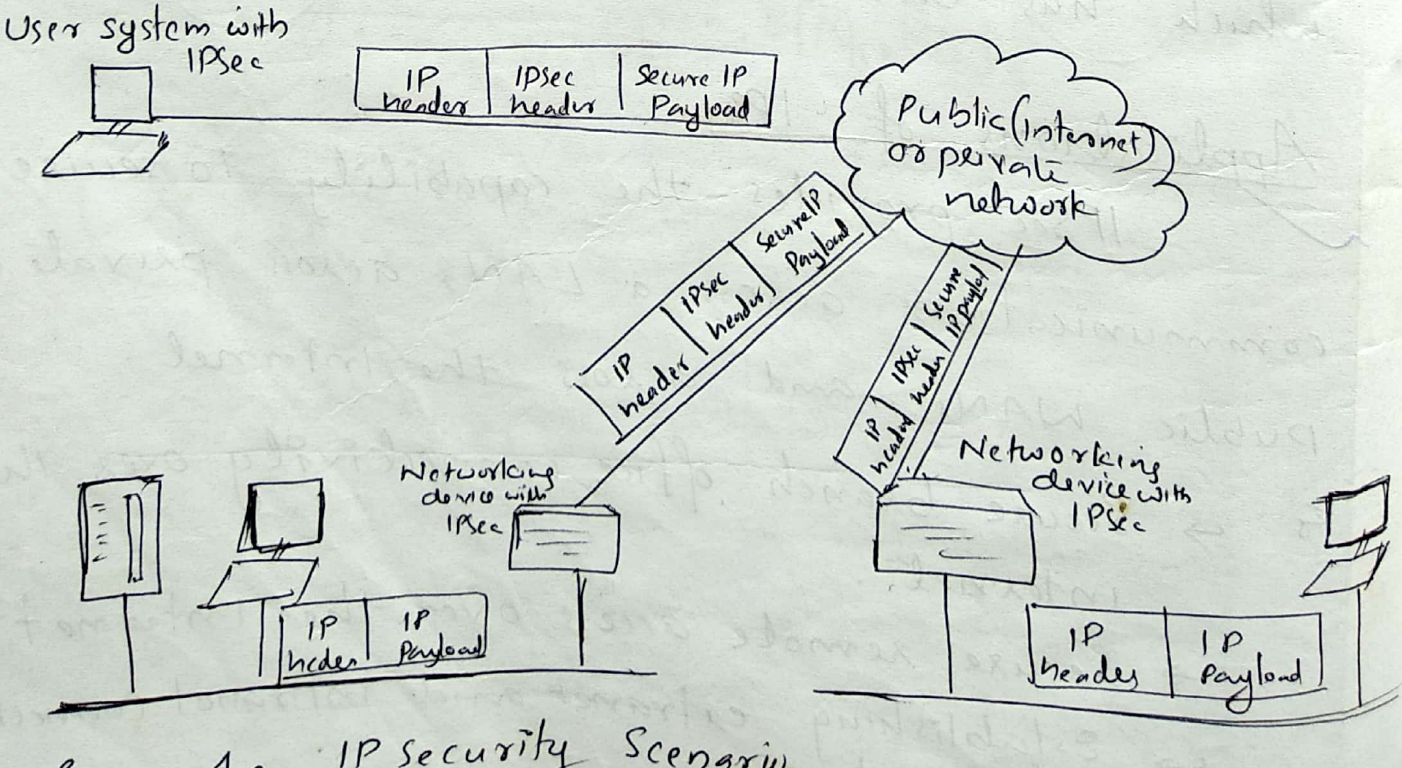
→ when IPSec is implemented in a firewall or router, it provides strong security. that can be applied to all traffic crossing the perimeter.

→ IPSec in a firewall is resistant to bypass if all traffic from the outside. must use IP, and the firewall is the only means of entrance from the internet into the organization.

→ IPSec is below the transport layer (TCP, UDP) and so is transparent to applications.

→ IPSec can be transparent to end users.

→ IPSec can provide security for individual users if needed.



fist An IP Security Scenario

An Organization maintains LANs at dispersed locations. Nonsecure IP traffic is conducted on each LAN. For traffic offsite, through some sort of private or public WAN, IPsec protocols are used. These protocols operate in networking devices, such as a router or firewall, that connect each LAN to the outside world. The IPsec networking device ~~with~~ will typically encrypt and compress all traffic going into the WAN, and decrypt and decompress traffic coming from the WAN. these operations are transparent to workstations and servers on the LAN. Secure transmissions are also possible with individual users who dial into the WAN. Such user workstations must implement the IPSec protocol to provide security.

② *-Imp*

IP Security Architecture

IPSec Documents

The IPSec Specifications consists of numerous documents. The most important documents are

→ RFC 2401 :- An overview of security architecture

→ RFC 2402 :- Description of a packet authentication extension to IPV4 and IPV6.

→ RFC 2406 :- Description of a packet encryption extension to IPV4 and IPV6.

→ RFC 2408 :- Specification of key management capabilities.

In addition to these 4 RFC's, no. of additional documents are published and are divided into 7 groups:

1. Architecture :- Covers the general concepts, security requirements, definitions and mechanisms defining IP sec technology.

*. 2. Encapsulating Security Payload (ESP) :- Covers the Packet format and general issues related to the use of the ESP for packet encryption.

3. Authentication Header (AH) : Covers the packet format and general issues related to the use of AH for packet authentication.

6 4. Encryption algorithm :- A set of documents that describes how various encryption algorithms are used for ESP.

5. Authentication Algorithm :- A set of documents that describe how various authentication algorithms are used for AH and for the authentication option of ESP.

6. Key Management :- Documents that describes key management schemes.

7. Domain of Interpretation (DOI) :- Contains values needed for the other documents to relate to each other.

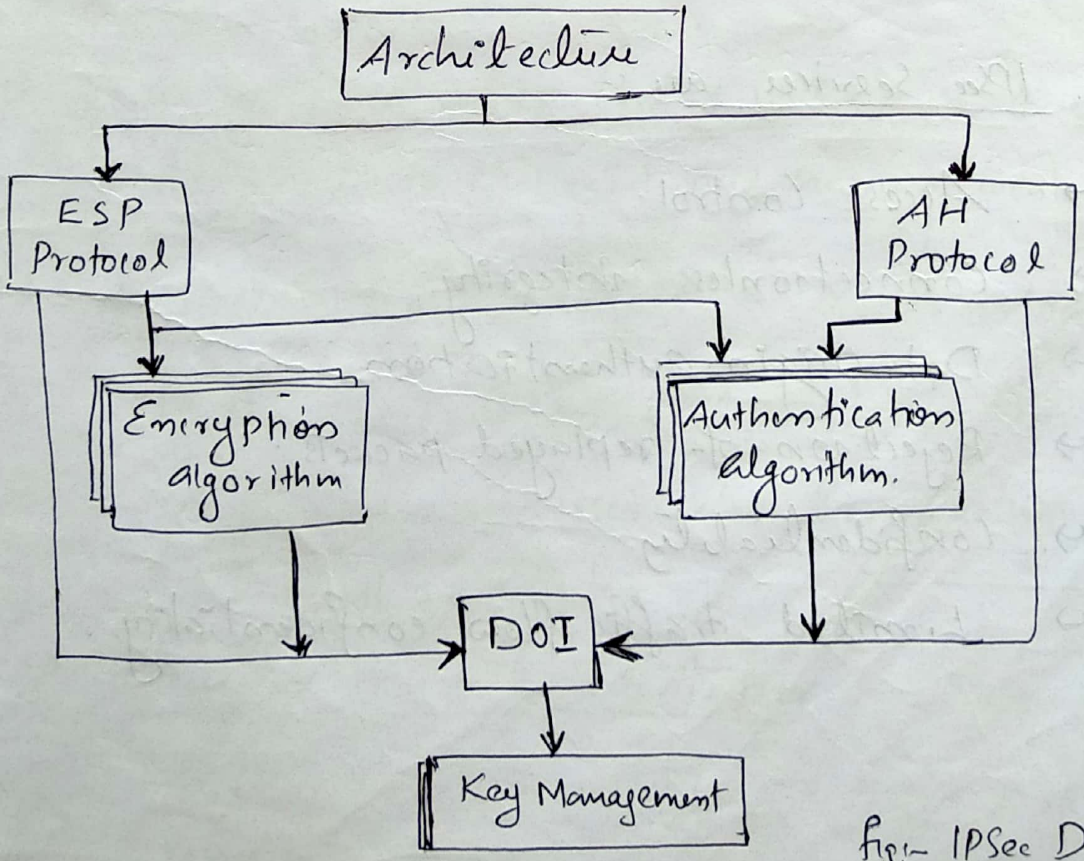


fig:- IPSec Document Overview.

6 **IPSec Services**

IPSec provides security services at the IP - layer by enabling a system to select required security protocols, determinine algorithms to use for the services, and put in place any cryplographic keys required to provide the requested serviles.

Two protocols are used to provide security:

1) Authentication protocol by Authentication header (AH)
2) Combined encryption/authentication protocol by Encapsulating Security Payload (ESP)

The IPSec Services are:-

- Access Control
- Connectionless integrity
- Data Origin authentication
- Rejection of replayed packets.
- Confidentiality
- Limited traffic flow confidentiality.

*important

Security Associations (SA)

1 A key concept that appears in both authentication and confidentiality mechanisms for IP Sec is the Security Associations (SA).

An <u>association is a one-way relationship between a sender and a receiver that affords security services to the traffic carried on it.</u>

if a peer relationship is needed, for two-way Secure exchange, then two security associations are required.

A security association is uniquely identified by 3 parameters :-

- <u>Security Parameter Index (SPI)</u> :- A bit string assigned to this SA and having local significance only
- <u>IP Destination address</u> :- ~~destination~~ address of the destination endpoint of the SA, which may be an end user stm or a ntwk systems such as firewall or router.
- <u>Security Protocol Identifier</u> :- This indicates whether the association is an AH or ESP Security associations.

SA Parameters

In each IPSec implementation, there is security association database that defines the parameters associated with each SA. Parameters are:-

→ Sequence Number Counter :- 32 bit value used to generate the sequence no field in AH or ESP header.

→ Sequence Counter Overflow :- A flag indicating the overflow of the sequence no. counter.

→ Anti-Replay window :- Used to determine whether an inbound AH or ESP packet is a replay.

→ ESP informations :- Encryption and authentication algorithms, keys, intialization values, key lifetime, and related parameters being used with ESP.

→ Lifetime of this Security Association :- A time interval or byte count after which an SA must be replaced with new SA.

→ IPSec Protocol Mode :- Tunnel or transport or wildcard mode.

→ Path MTU :- Any observed path for maximum transmission unit and aging variables.

a

SA Selectors:

The means by which IP traffic is related to specific SAs is the nominal Security Policy Database (SPD). An SPD contain entrie each of which defnies a subset of IP traffic and points to an SA for that traffic.

Each SPD entry is defined by a set of IP and upper-layer protocol field-values, called selectors.

Selectors are :-

- Destination IP address.
- Source IP address
- User ID
- Data sensitivity level
- Transport layer protocol
- IPSec Protocol (AH or ESP)
- Source and destination port
- IPV6 class
- IPv6 flow label
- IPV4 Type of service

X. important

10

Transport and Tunnel Mode

Both AH and ESP support two modes of operation: Transport and tunnel mode.

Transport Mode

Transport mode provides protection primarily for upper-layer protocols. ie transport mode protection extends to the payload of an IP packet. Transport mode is used for end-to-end communication between two hosts.

ESP in transport mode encrypts and optionally authenticates the IP payload but not the IP header.

AH in transport mode authenticate the IP payload and selected portion of the IP header.

Tunnel Mode

Tunnel mode provides protection to the entire IP packet. Tunnel mode is used when one or both ends of a SA is a security gateway, such as a firewall or router that implements IPSec.

ESP in tunnel mode encrypts and optionally authenticate the entire inner IP packet, including the inner IP header.

AH in tunnel mode authenticates the entire inner IP packet and selected portion of the outer IP header.

③ ‖ Authentication Header (AH)

*** important.

AH provides support for data integrity and authentication of IP packets. The data integrity features ensures that undetected modification to a packet's content in transit is not possible. The authentication feature enables an end system or network device to authenticate the user or application and filter traffic accordingly. It also prevents the address spoofing attacks and replay attacks.

Bit : 0	8	16 31
Next header	Payload length	Reserved
Security Parameters index (SPI)		
Sequence Number		
Authentication data (variable)		

fig- IPSec Authentication Header.

Fields of AH are :-

→ Next header (8 bits) :- Identifies the type of header immediatly following this header.

→ Payload Length (8 bits) :- Length of AH in 32 bit word minus 2.

12 → Reserved (16 bits) :- for future use.

→ Security Parameter Index (SPI) (32 bits) :- Identifi the security associations.

→ Sequence number (32 bits) :- An increasing counter value.

→ Authentication data (variable) :- A variable length field (integral no. of 32 bit) that contains the Integrity check Value (ICV) or MAC for this packet.

Anti-Replay Service

A replay attack is one which an attacker obtains a copy of an authenticated packet and later transmit it to the intended destination. The receipt of duplicate authenticated IP Packets may disrupt service in some way or may have some other undesired consequence. The sequence number field is used to prevent such attack.

Sequence Number generation and Processing is as follows:-

when a new SA is established, the sender initializes a sequence number counter

Each time that a packet is sent on this SA the sender increments the counter and places the value in the sequence number field. Thus the first value to be used is 1. If the limit of $2^{32}-1$ is reached, the sender should terminate this SA and negotiate a new SA with a new key.

Because IP is connectionless, unreliable - service, the protocol does not gusrantee that packets will be delivered in order and does not guarantee that all packets will be delivered. Therefore, the IPsec authentication document dictates that the receiver should implement a window of size W, with a default of W = 64. ~~The~~

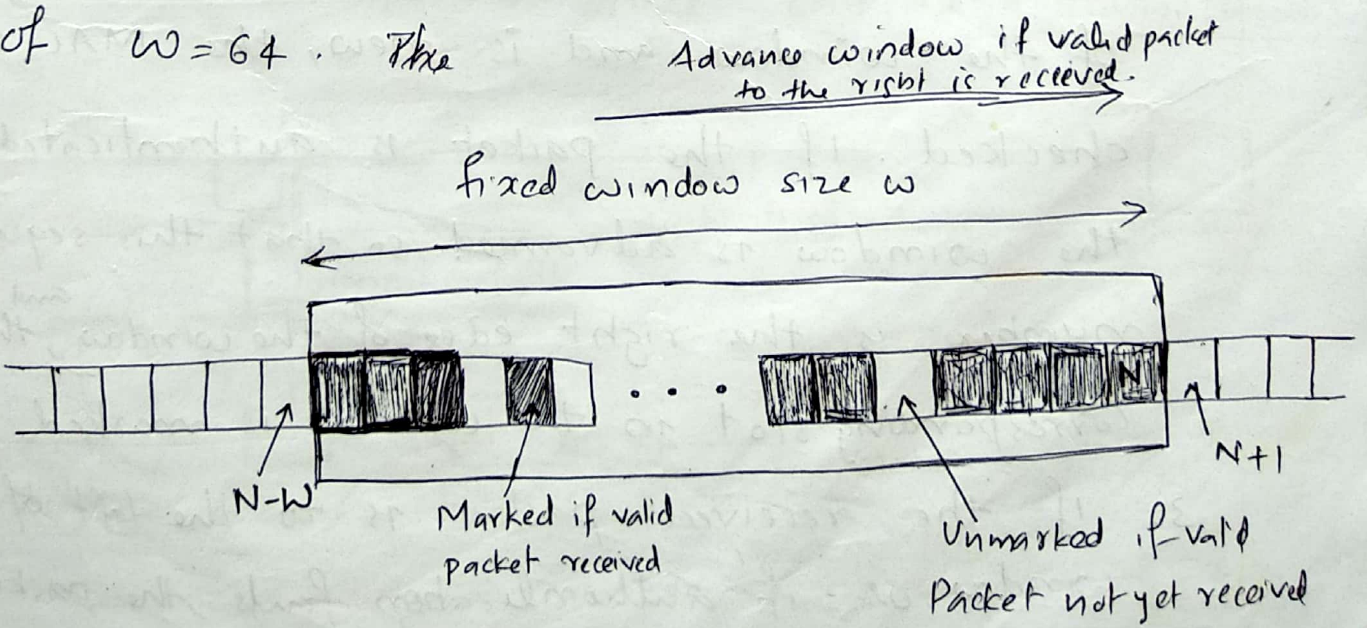

Advance window if valid packet to the right is received.
fixed window size W
N
N-W
N+1
Marked if valid packet received
Unmarked if valid Packet not yet received

fig:- Anti Replay Mechanism.

The right edge of the window represents the highest sequence number, N, so far received for a valid packet. For any packet with a sequence number in the range from N-W+1 to N that has been correctly received, the corresponding slot in the window is mark.

The processing is as follows:-

1. if the received packet falls within the window and is new, the MAC is checked. if the packet is authenticated, the corresponding slot in the window is marked.

2. if the received packet is to the right of the window and is new, the MAC is checked. If the packet is authenticated, the window is advanced so that this sequence number is the right edge of the window, and the corresponding slot in the window is marked

3. if the received packet is to the left of the window, or if authentication fails, the packet is discarded, this is an auditable event,

Transport and Tunnel Modes

There are two ways in which the IPSec authentication service can be used.

- Authentication is provided directly btwo a Server and client workstations. the workstations can be either on the same network as the server or on an external network. As long as the workstation and the server share a protected secret key, the authentication process is secure. This case uses a transport mode SA
- A remote workstation authenticate itself to the corporate firewall, either for access to the entire internal network or because the requested server does not support the authentication feature. This case use a tunnel mode SA

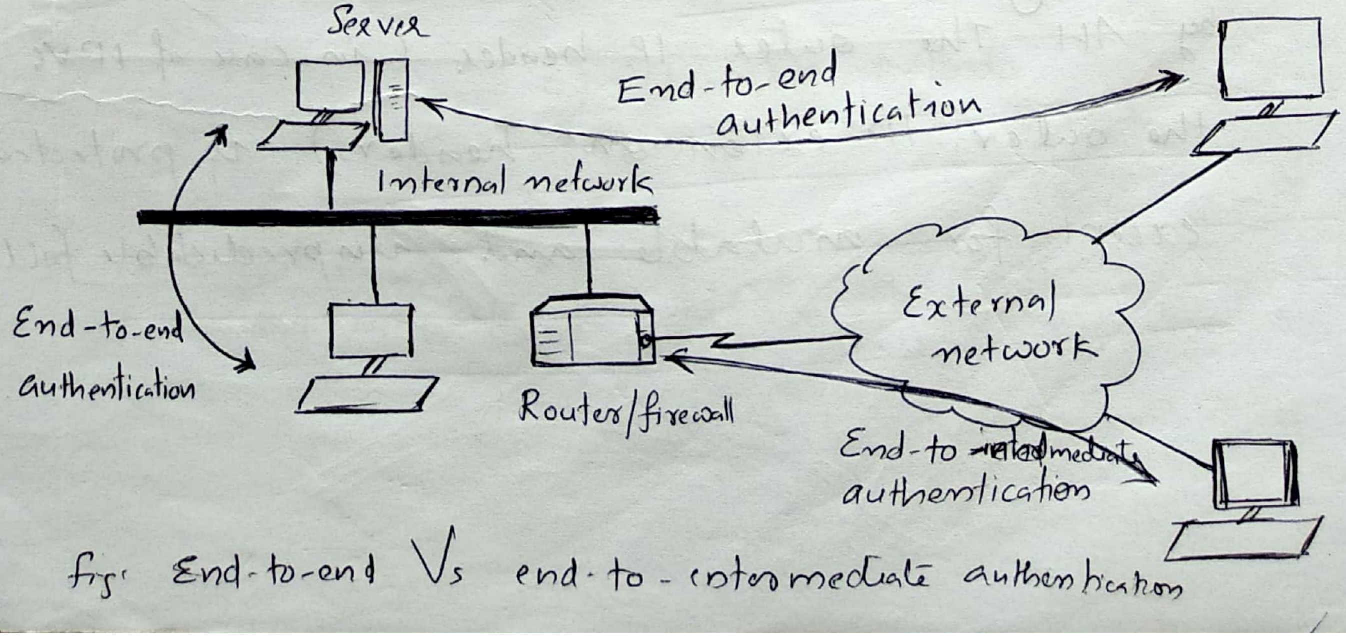
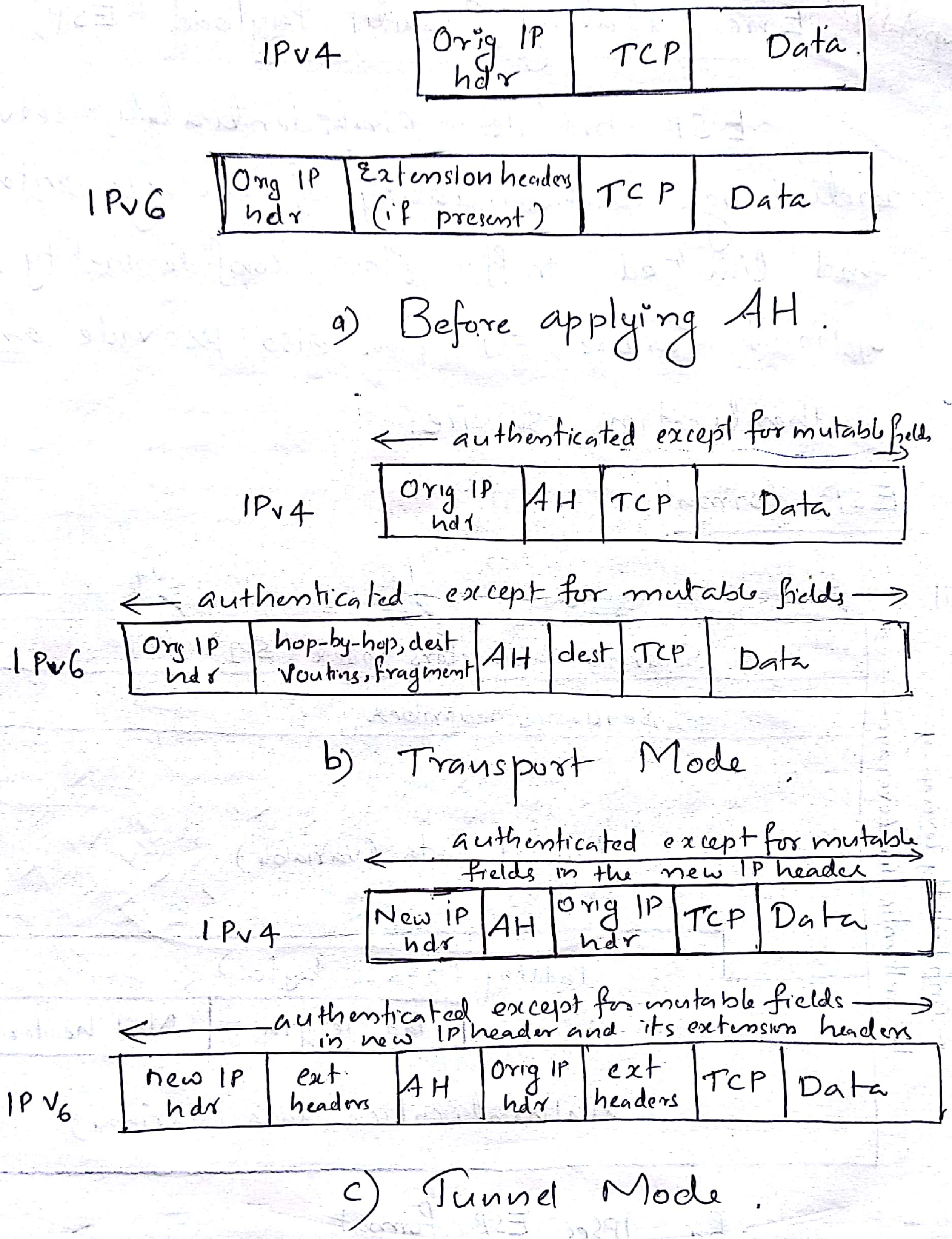

Server
End-to-end authentication
Internal network
End-to-end authentication
External network
Router/firewall
End-to intermediate authentication

fig: End-to-end Vs end-to-intermediate authentication

Scope of AH authentication

For transport mode AH using IPv4, the AH is inserted after the original IP header and before the IP payload. In IPV6, AH is viewed as an end-to-end pay load. ie it is not examined or processed by intermediate routers. therefore AH appears after the IPV6 base header and the hop-by-hop, routing and fragment extension headers.

For tunnel mode AH, the entire original IP packet is authenticated, and the AH is inserted between the original IP header and a new outer IP header.
with tunnel mode, the entire inner IP packet, including the entire inner IP header is protected by AH. The outer IP header (in case of IPV6 the outer IP extension headers) is protected except for mutable and unpredictable fields.


IPv4
Orig IP hdr
TCP
Data
IPv6
Orig IP hdr
Extension headers (if present)
TCP
Data
authenticated except for mutable fields
IPv4
Orig IP hdr
AH
TCP
Data
authenticated except for mutable fields
IPv6
Orig IP hdr
hop-by-hop, dest routing, fragment
AH
dest
TCP
Data
authenticated except for mutable fields in the new IP header
IPv4
New IP hdr
AH
Orig IP hdr
TCP
Data
authenticated except for mutable fields in new IP header and its extension headers
IPV6
new IP hdr
ext headers
AH
Orig IP hdr
ext headers
TCP
Data

a) Before applying AH.

b) Transport Mode.

c) Tunnel Mode.

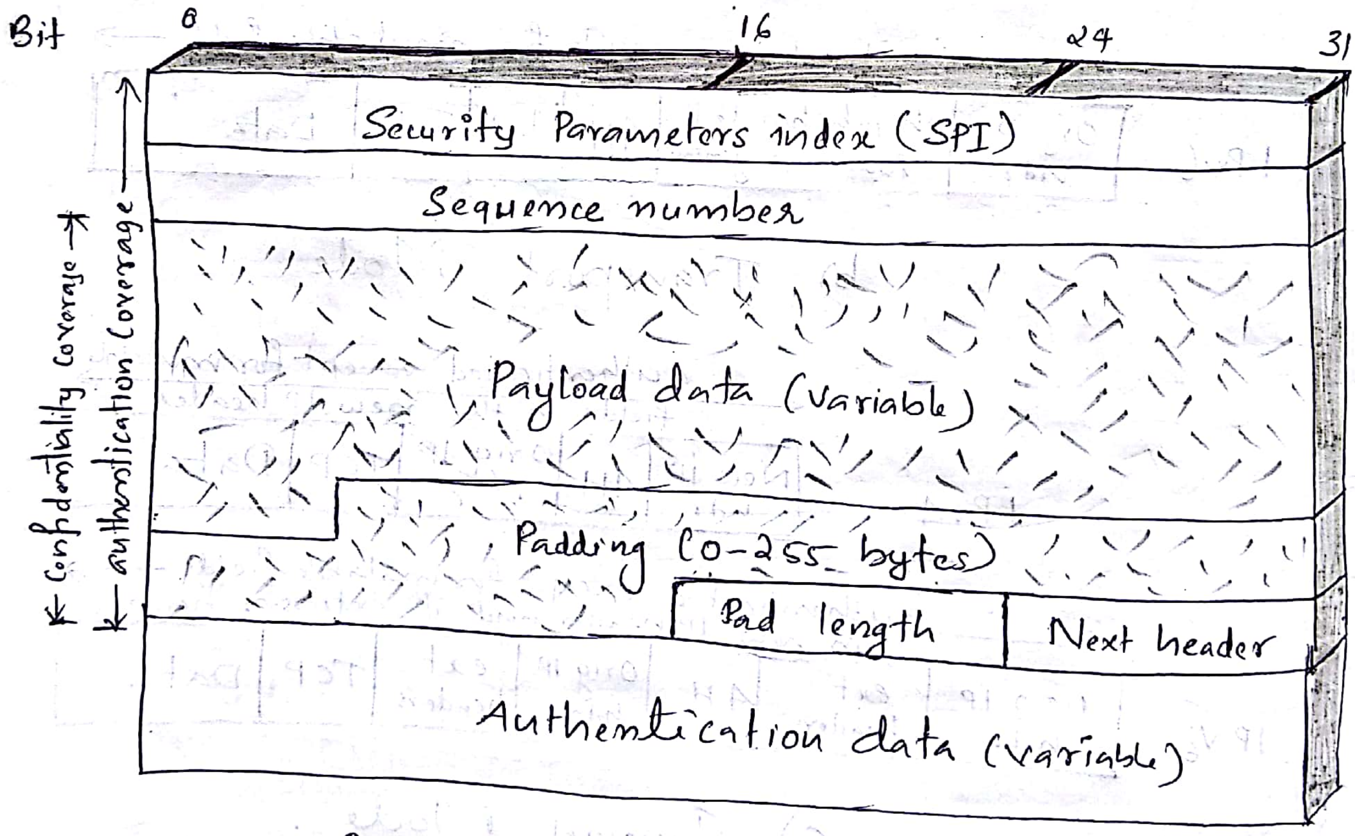
Fig:- Scope of AH authentication.

:X:X important (4)

Encapsulating Security Payload (ESP)

13 ESP provides confidentiality services, including confidentiality of message contents and limited traffic flow confidentiality. As an optional feature, ESP can also provide an authentication service.

ESP format



Bit
0
16
24
31
Security Parameters index (SPI)
Sequence number
Payload data (variable)
Padding (0-255 bytes)
Pad length
Next header
Authentication data (variable)
← confidentiality coverage →
← authentication coverage →

fig IPSec ESP format

ESP packets contains the following fields :-

→ Security Parameter Index (32 bits) :- identifies a SA

→ Sequence Number (32 bits) : An increasing counter value.

→ Payload Data (variable) :- This is a transport-level segment or IP packet that is protected by encryption.

14 → Padding (0-255 bytes) :-

→ Pad length (8 bits) :- Indicates the no. of pad bytes immediatly preceeding this field.

→ Next header (8 bits) :- Identifies the type of data contained in the payload data field by identifying the first header in that payload.

→ Authentication Data (variable) :- It contains the integrity check value computed over the ESP Packet minus the Authentication Data field.

Encryption and authentication algorithms.

Encryption algorithms :-

→ DES in CBC mode
→ 3-key triple DES
→ RC5
→ IDEA
→ CAST
→ Blowfish.

authentication algorithms are :-

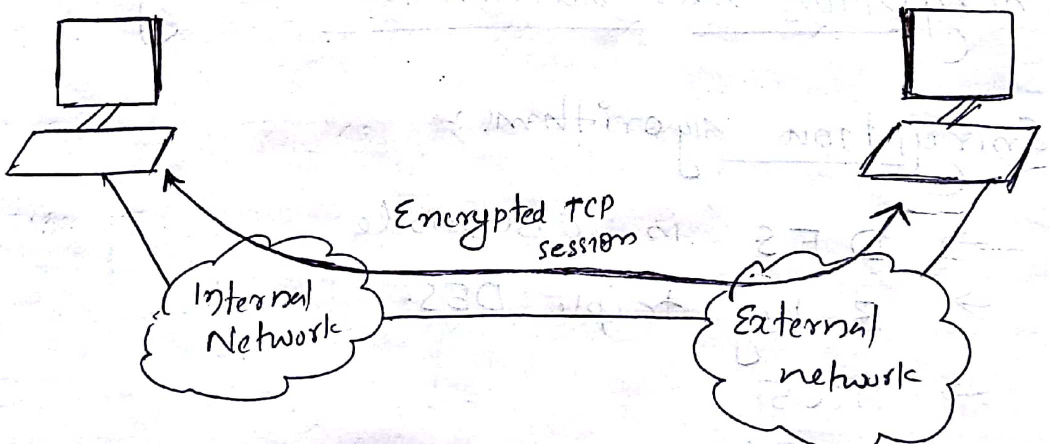
→ MAC with a default length of 96 bits.
→ HMAC-MD5-96.
→ HMAC-SHA-1-96.

15 Transport and Tunnel Mode

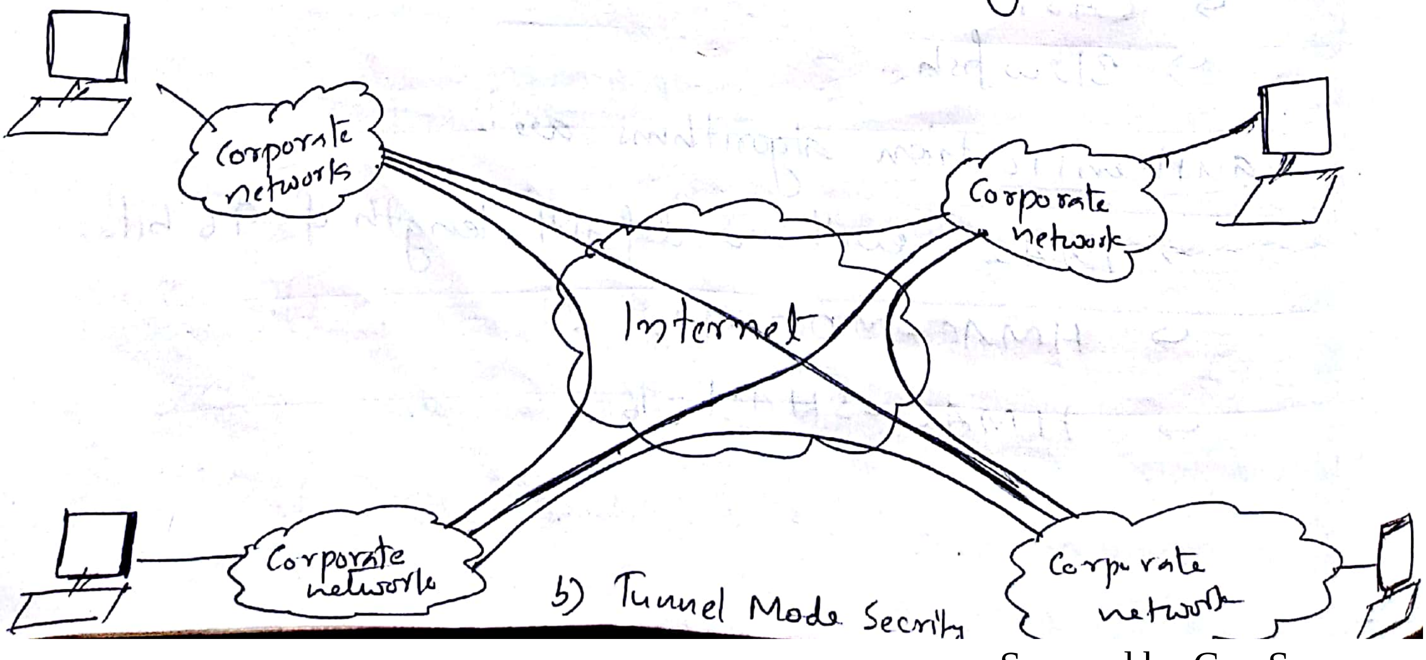
There are 2 ways in which the IPSec ESP service can be use.

→ Encryption is provided directly between two hosts. It is supported by transport mode SA.

↳ Hosts on the internal networks use the Internet for transport of data but do not interact with other-internet based hosts. It is supported by tunnel mode SA. It is used to set up a virtual private network

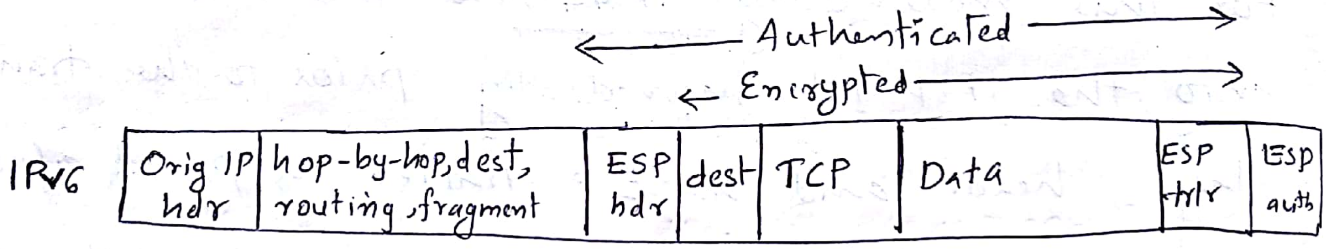


a) Transport level Security



b) Tunnel Mode Secrity

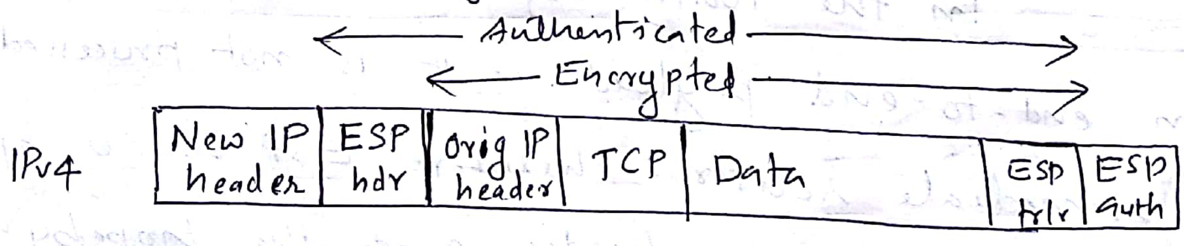
plus the destination options extension header if it occurs after the ESP header. Again the authentication covers the ciphestext plus the ESP header.



Tunnel Mode ESP

Tunnel mode ESP is used to encrypt an entire IP packet. For this mode, the ESP header is prefixed to the packet and then the packet plus the ESP trailer is encrypted.

For IPv4 and IPv6 tunnel mode encryption and Authenticatn is as follows:-



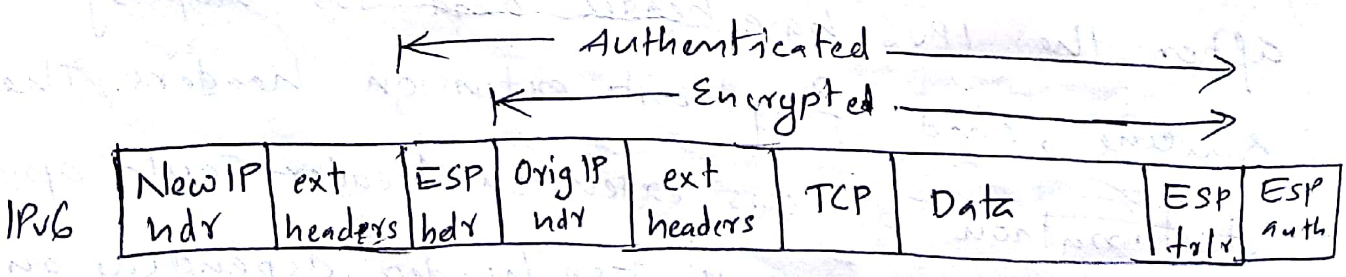
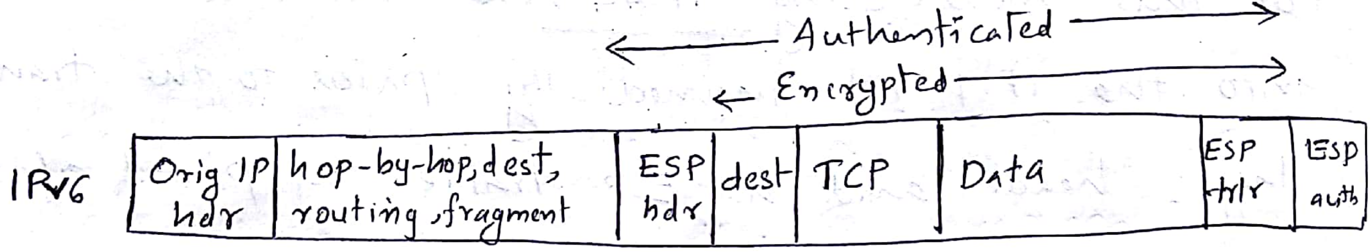


fig:- Tunnel Mode.

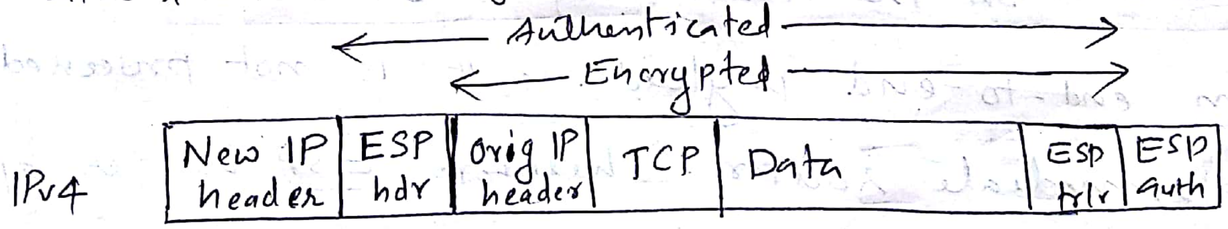
plus the destination optrous extension header if it occurs after the ESP header. Again the authentication covers the ciphertext plus the ESP header.



Tunnel Mode ESP

Tunnel mode ESP is used to encrypt an entire IP packet. For this mode, the ESP header is prefixed to the packet and then the packet plus the ESP trailer is encrypted.

For IPv4 and IPVc tunnel mode encryption and authenticatn is as follows:-



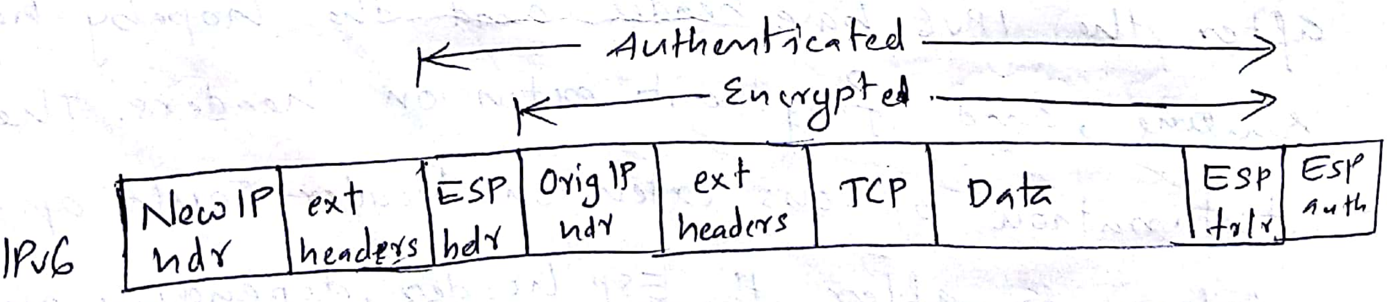


fig:- Tunnel Mode.

(5) Combining Security Associations

18 An individual SA can implement either the AH or ESP protocol but not both. Sometimes a particular traffic flow will call for the services provided by both AH and ESP.

Security associations can be combined into bundles into two ways.

1) Transport adjacency : Refers to applying more than one security protocol to the same IP packet without invoking tunneling.

2) Iterated tunneling :- Refers to the applicatn of multiple layers of security protocols effected through IP tunneling.

Basic Combinations of SA

The IPSec Architecture lists 4 examples of Combinations of SAs.

Case 1 : Here all security is provided between end Systems that implement IPSec. For any two end Systems to communicate via an SA there must share the appropriate secret key.

Possible Combinatns are :-

a) AH in transport mode

b) ESP in transport mode

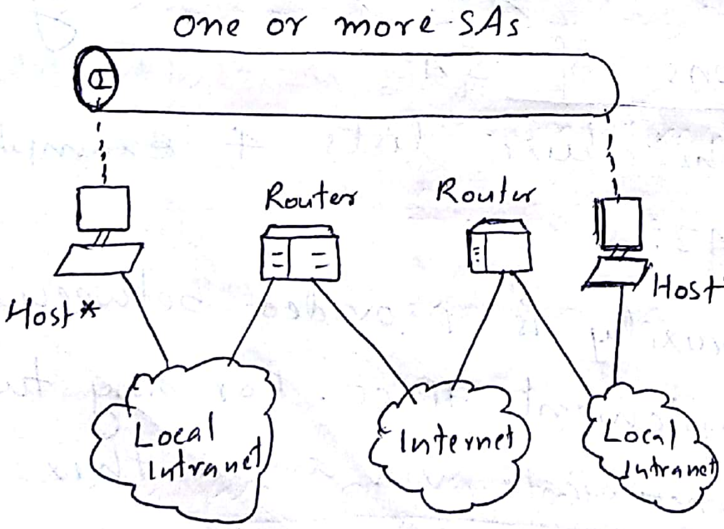
c) AH followed by ESP in transport mode

d) any of a,b,c inside AH or ES in tunnel mode.

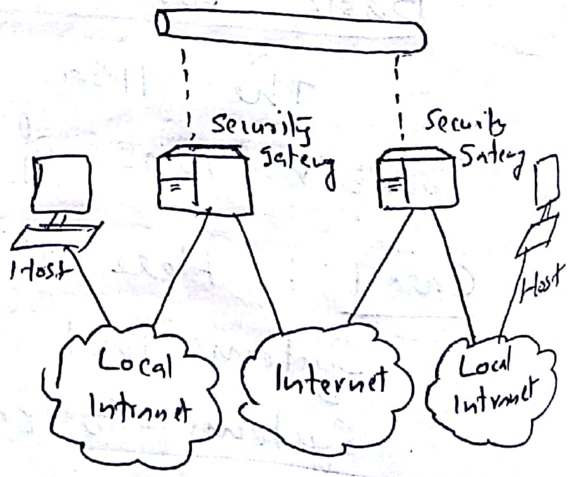
Case 2 : Here the security is provided only between gateways and no hosts implement IPSec. ~~The~~ A Single tunnel ~~could~~ SA only needed in this case. The tunnel could support AH, ESP or ESP with authentication option.

Case 3 : It build on case 2 by adding end-to-end security. All combinations for case 1 and 2 are allowed here.

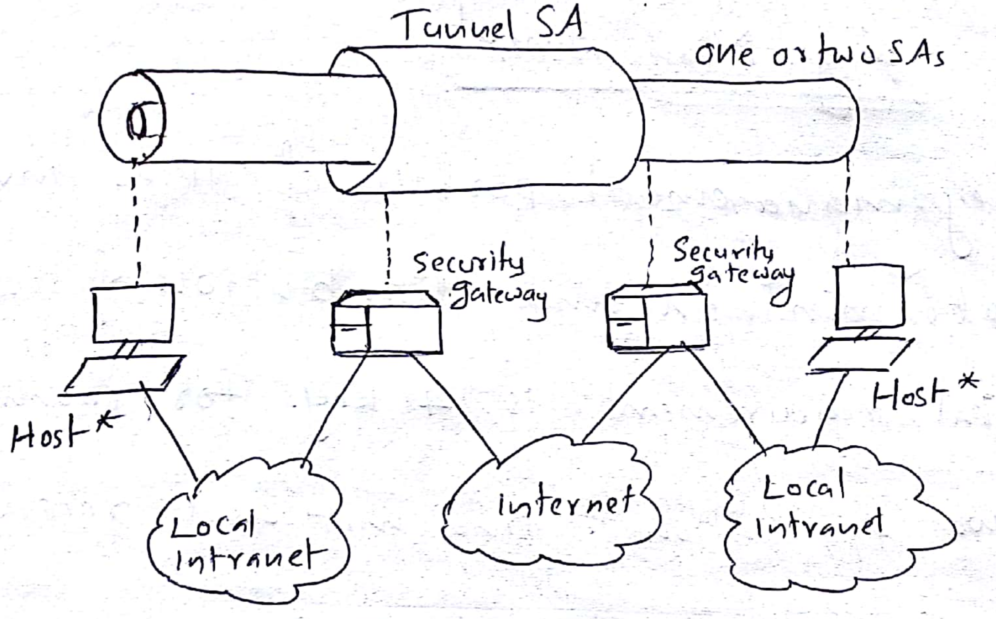
Case 4 : It provides support for a remote host that uses the Internet to reach an organization's firewall and then to gain access to Some server or workstation behind the firewall. Only tunnel mode is required.



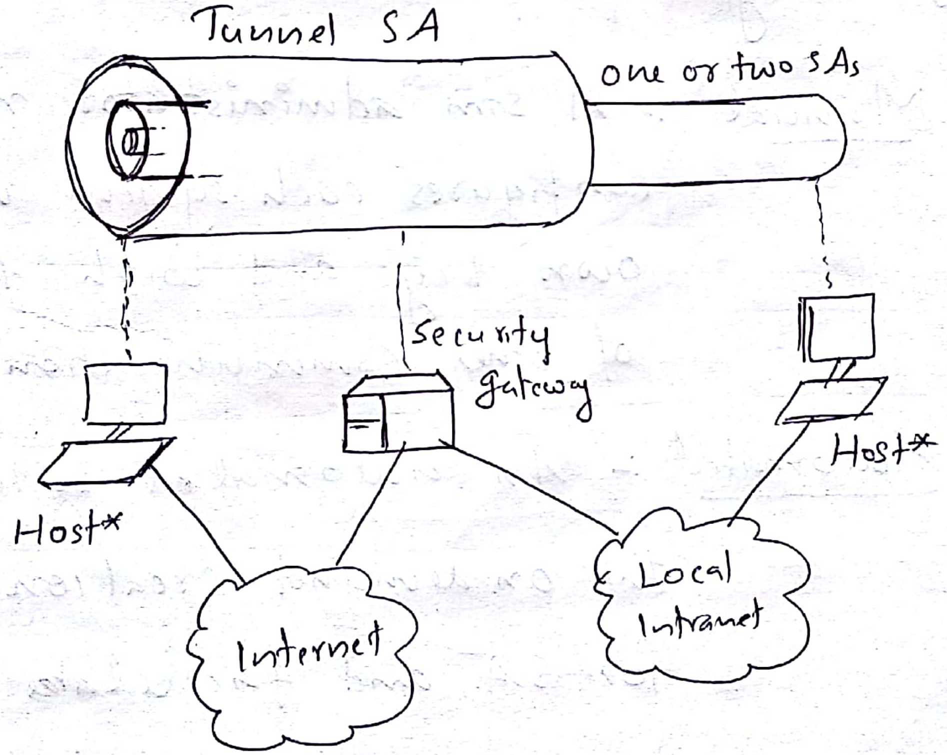
a) case 1



b) Case 2.



C) Case (3)



d) Case 4

* - denote Host that implements IPSec.

fig:- Basic Combinations of Security Associations.

(6) Key Management

Key management portion of IPSec involves the determination and distribution of secret keys. A typical requirement is 4 keys for communicatn btw two applications : transmit and receive pairs for both AH and ESP.

IPSec architecture documents support 2 types of key management.

→ Manual :- A stm administrator manually configures each system with its own keys and with the keys of other communication systems.

→ Automated :- An automated system enables the on-demand creation of keys for SAs and facilitates the use of keys in a large distributed system with an evolving configuration.

There are 2 automated key management protocols

→ Oakley key Determination protocol

→ Internet Security Association and Key Management protocol (ISAKMP)

UNIT-VI: WEB SECURITY

Web Security Considerations, Secure Socket Layer (SSL) and Transport Layer Security (TLS), Secure Electronic Transaction (SET).

Introduction:

Virtually all businesses, most government agencies, and many individuals now have Web sites. The number of individuals and companies with Internet access is expanding rapidly and all of these have graphical Web browsers. As a result, businesses are enthusiastic about setting up facilities on the Web for electronic commerce. But the reality is that the Internet and the Web are extremely vulnerable to compromises of various sorts. As businesses wake up to this reality, the demand for secure Web services grows.

6.1 Web Security Considerations:

The World Wide Web is fundamentally a client/server application running over the Internet and TCP/IP intranets. the Web presents new challenges not generally appreciated in the context of computer and network security.

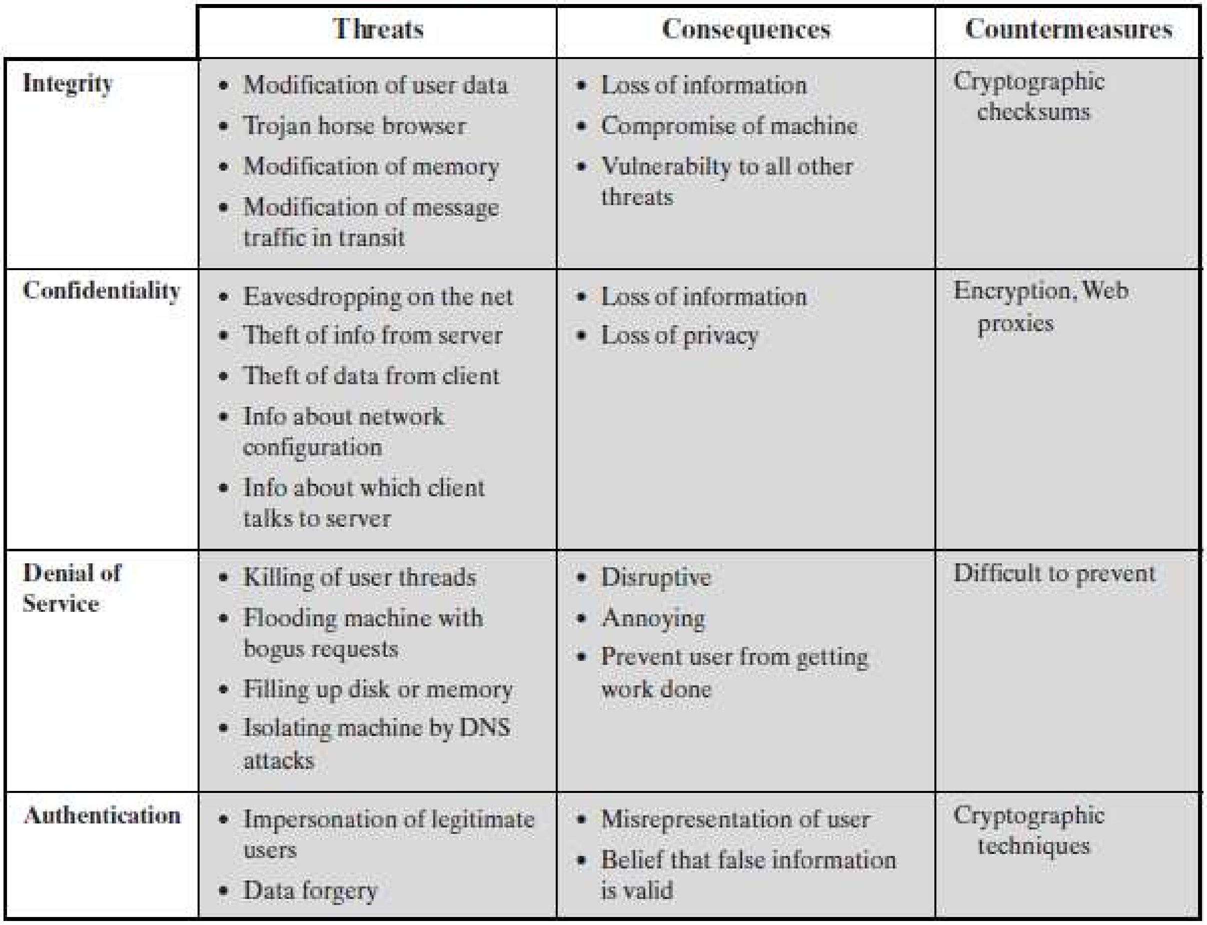
- The Internet is two-way. Unlike traditional publishing environments—even electronic publishing systems involving teletext, voice response, or fax-back— the Web is vulnerable to attacks on the Web servers over the Internet.
- The Web is increasingly serving as a highly visible outlet for corporate and product information and as the platform for business transactions. Reputations can be damaged and money can be lost if the Web servers are subverted.
- Although Web browsers are very easy to use, Web servers are relatively easy to configure and manage, and Web content is increasingly easy to develop, the underlying software is extraordinarily complex. This complex software may hide many potential security flaws. The short history of the Web is filled with examples of new and upgraded systems, properly installed, that are vulnerable to a variety of security attacks.
- A Web server can be exploited as a launching pad into the corporation’s or agency’s entire computer complex. Once the Web server is subverted, an attacker may be able to gain access to data and systems not part of the Web itself but connected to the server at the local site.
- Casual and untrained (in security matters) users are common clients for Web-based services. Such users are not necessarily aware of the security risks that exist and do not have the tools or knowledge to take effective countermeasures.

6.1.1Web Security Threats

- Web now widely used by business, government, individuals
- but Internet & Web are vulnerable
- have a variety of threats

Table below provides a summary of the types of security threats faced when using the Web.

Table 5.1 A Comparison of Threats on the Web



	Threats	Consequences	Countermeasures
Integrity	• Modification of user data • Trojan horse browser • Modification of memory • Modification of message traffic in transit	• Loss of information • Compromise of machine • Vulnerabilty to all other threats	Cryptographic checksums
Confidentiality	• Eavesdropping on the net • Theft of info from server • Theft of data from client • Info about network configuration • Info about which client talks to server	• Loss of information • Loss of privacy	Encryption, Web proxies
Denial of Service	• Killing of user threads • Flooding machine with bogus requests • Filling up disk or memory • Isolating machine by DNS attacks	• Disruptive • Annoying • Prevent user from getting work done	Difficult to prevent
Authentication	• Impersonation of legitimate users • Data forgery	• Misrepresentation of user • Belief that false information is valid	Cryptographic techniques

6.1.2: Web Traffic Security Approaches

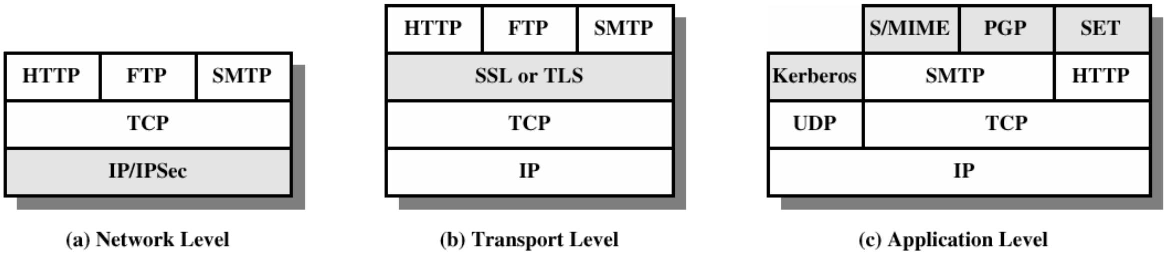
A number of approaches to providing Web security are possible. The various approaches that have been considered are similar in the services they provide and, to some extent, in the mechanisms that they use, but they differ with respect to their scope of applicability and their relative location within the TCP/IP protocol stack.

Figure 5.1 illustrates this difference. One way to provide Web security is to use IP security (IPsec) (Figure 5.1a).The advantage of using IPsec is that it is transparent to end users and applications and provides a general-purpose solution. Furthermore, IPsec includes a filtering capability so that only selected traffic need incur the overhead of IPsec processing.

Another relatively general-purpose solution is to implement security just above TCP (Figure 5.1b). The foremost example of this approach is the Secure Sockets Layer (SSL) and the follow-on Internet standard known as Transport Layer Security (TLS). At this level, there are two implementation choices. For full generality, SSL (or TLS) could be provided as part of the underlying protocol suite and therefore be transparent to applications. Alternatively, SSL can be embedded in specific packages. For example, Netscape and Microsoft Explorer browsers come equipped with SSL, and most Web servers have implemented the protocol.

Application-specific security services are embedded within the particular application.

Figure 5.1c shows examples of this architecture. The advantage of this approach is that the service can be tailored to the specific needs of a given application.



6.2 SECURE SOCKET LAYER (SSL) AND TRANSPORT LAYER SECURITY (TLS):

- SSL – Secure Socket Layer
- TLS – Transport Layer Security
- both provide a secure transport connection between applications (e.g., a web server and a browser)
- SSL was developed by Netscape
- SSL version 3.0 has been implemented in many web browsers (e.g., Netscape Navigator and MS Internet Explorer) and web servers and widely used on the Internet
- SSL v3.0 was specified in an Internet Draft (1996)
- it evolved into TLS specified in RFC 2246
- TLS can be viewed as SSL v3.1

6.2.1 SSL Architecture

- SSL is designed to make use of TCP to provide a reliable end-to-end secure service.
- SSL is not a single protocol but rather two layers of protocols.
- Two important SSL concepts are the SSL session and the SSL connection, which are defined in the specification as follows.

- **Session**: An SSL session is an association between a client and a server. Sessions are created by the Handshake Protocol. Sessions define a set of cryptographic security parameters which can be shared among multiple connections. Sessions are used to avoid the expensive negotiation of new security parameters for each connection.
- **Connection**: A connection is a transport that provides a suitable type of service. For SSL, such connections are peer-to-peer relationships. The connections are transient. Every connection is associated with one session.
-
- A **session state** is defined by the following parameters.
- **Session identifier**: An arbitrary byte sequence chosen by the server to identify an active or resumable session state.
- **Peer certificate:** An X509.v3 certificate of the peer. This element of the state may be null.
- **Compression method:** The algorithm used to compress data prior to encryption.
- **Cipher spec:** Specifies the bulk data encryption algorithm (such as null,AES, etc.) and a hash algorithm (such as MD5 or SHA-1) used for MAC calculation. It also defines cryptographic attributes such as the hash_size.
- **Master secret**: 48-byte secret shared between the client and server.

- **Is resumable**: A flag indicating whether the session can be used to initiate new connections.
-
- A **connection state** is defined by the following parameters.
- **Server and client random**: Byte sequences that are chosen by the server and client for each connection.
- **Server write MAC secret:** The secret key used in MAC operations on data sent by the server.
- **Client write MAC secret:** The secret key used in MAC operations on data sent by the client.
- **Server write key**: The secret encryption key for data encrypted by the server and decrypted by the client.
- **Client write key**: The symmetric encryption key for data encrypted by the client and decrypted by the server.
- **Initialization vectors**: When a block cipher in CBC mode is used, an initialization vector (IV) is maintained for each key. This field is first initialized by the SSL Handshake Protocol. Thereafter, the final ciphertext block from each record is preserved for use as the IV with the following record.
- **Sequence numbers:** Each party maintains separate sequence numbers for transmitted and received messages for each connection. When a party sends or receives a change cipher spec message, the appropriate sequence number is set to zero. Sequence numbers may not exceed $2^{64} - 1$.

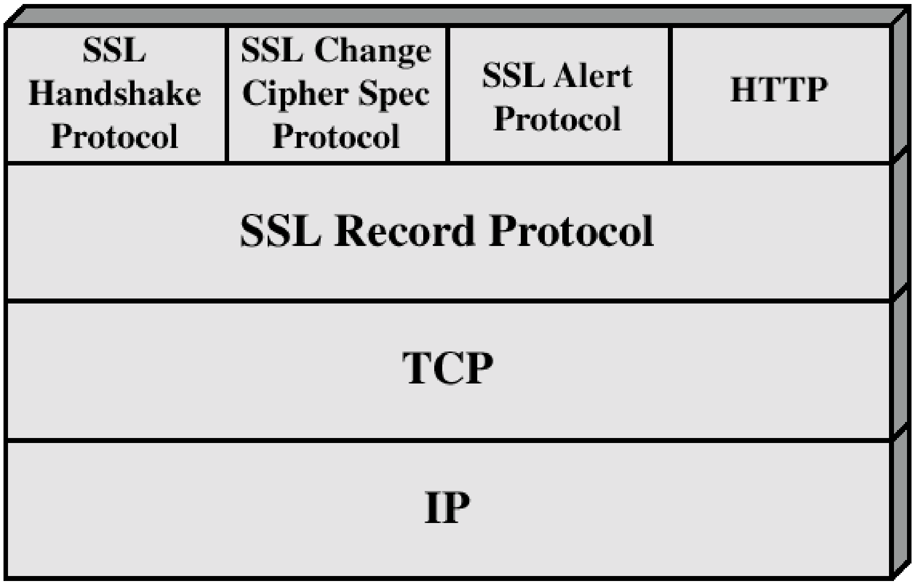


Figure 7.2 SSL Protocol Stack

SSL components:

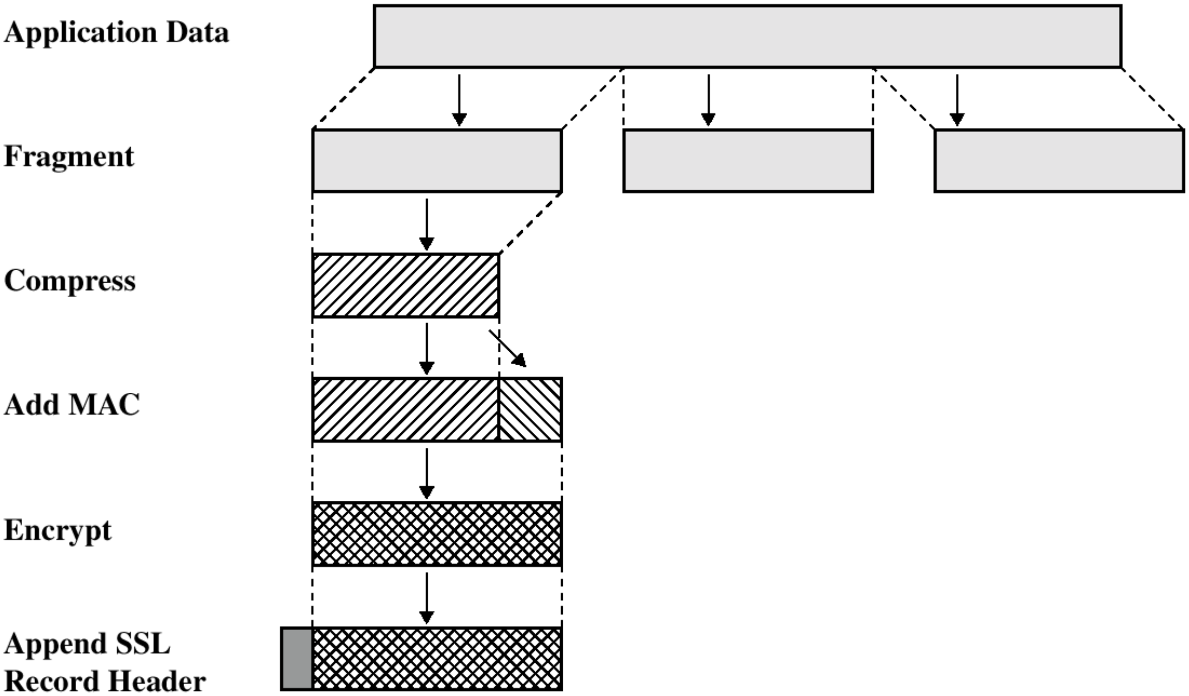
- SSL Handshake Protocol
  - negotiation of security algorithms and parameters
  - key exchange
  - server authentication and optionally client authentication
- SSL Record Protocol

    - fragmentation
    - compression
    - message authentication and integrity protection
    - encryption
- SSL Alert Protocol
    - error messages (fatal alerts and warnings)
- SSL Change Cipher Spec Protocol
    - a single message that indicates the end of the SSL handshake

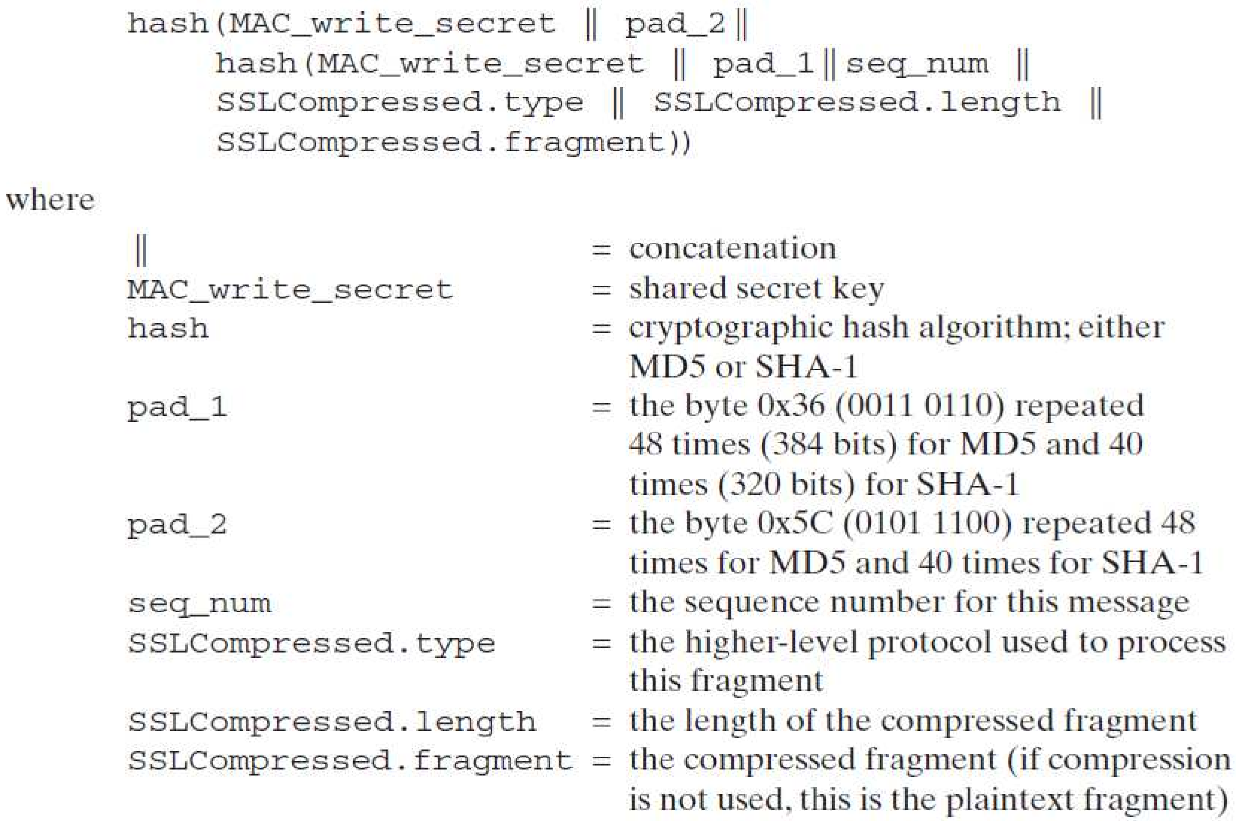
SSL Record Protocol:

- The SSL Record Protocol provides two services for SSL connections:
    - Confidentiality: The Handshake Protocol defines a shared secret key that is used for conventional encryption of SSL payloads.
    - Message Integrity: The Handshake Protocol also defines a shared secret key that is used to form a message authentication code(MAC).

SSL Record Protocol Operation:



- The first step is **fragmentation**.
- Each upper-layer message is fragmented into blocks of 2^{14} bytes (16384 bytes) or less. Next, compression is optionally applied.
- **Compression** must be lossless and may not increase the content length by more than 1024 bytes.
- In SSLv3 (as well as the current version of TLS), no compression algorithm is specified, so the default compression algorithm is null.
- The next step in processing is to compute a **message authentication code over** the compressed data. For this purpose, a shared secret key is used.
- The calculation is defined as.



```
hash(MAC_write_secret || pad_2||
    hash(MAC_write_secret || pad_1||seq_num ||
    SSLCompressed.type || SSLCompressed.length ||
    SSLCompressed.fragment))
```

where

|| = concatenation

`MAC_write_secret` = shared secret key

`hash` = cryptographic hash algorithm; either MD5 or SHA-1

`pad_1` = the byte 0x36 (0011 0110) repeated 48 times (384 bits) for MD5 and 40 times (320 bits) for SHA-1

`pad_2` = the byte 0x5C (0101 1100) repeated 48 times for MD5 and 40 times for SHA-1

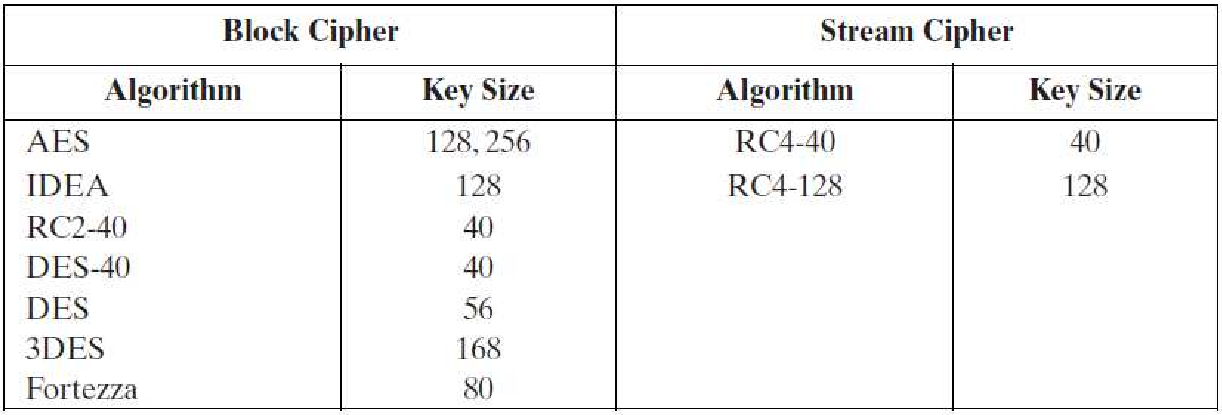
`seq_num` = the sequence number for this message

`SSLCompressed.type` = the higher-level protocol used to process this fragment

`SSLCompressed.length` = the length of the compressed fragment

`SSLCompressed.fragment` = the compressed fragment (if compression is not used, this is the plaintext fragment)

- Next, the **compressed message plus the MAC** are encrypted using symmetric encryption.
- Encryption may not increase the content length by more than 1024 bytes, so that the total length may not exceed 2^{14} + 2048.
- The following encryption algorithms are permitted:

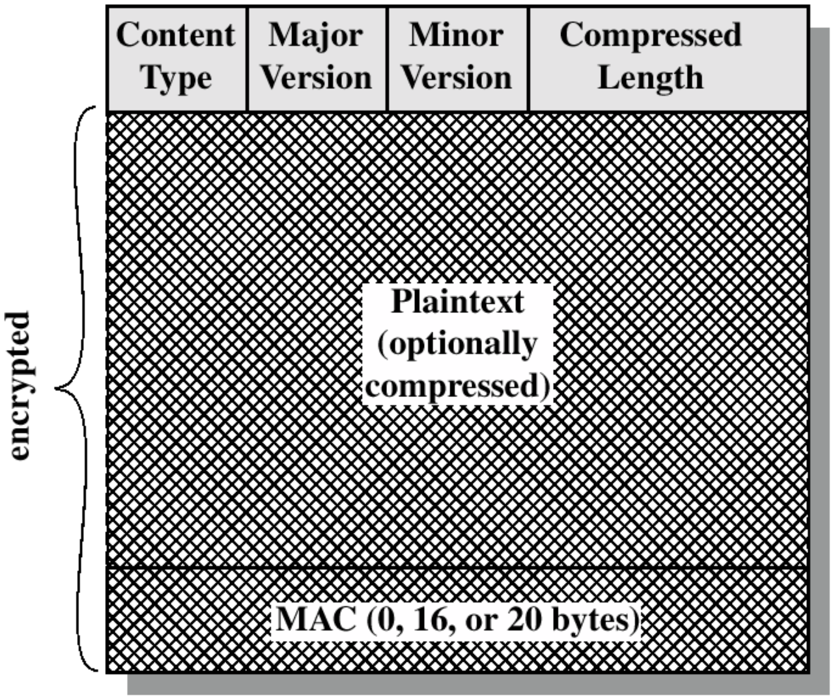


Block Cipher		Stream Cipher	
Algorithm	**Key Size**	**Algorithm**	**Key Size**
AES	128, 256	RC4-40	40
IDEA	128	RC4-128	128
RC2-40	40		
DES-40	40		
DES	56		
3DES	168		
Fortezza	80		

- Fortezza can be used in a smart card encryption scheme.
- For stream encryption, the compressed message plus the MAC are encrypted.
- Note that the MAC is computed before encryption takes place and that the MAC is then encrypted along with the plaintext or compressed plaintext.
- For block encryption, padding may be added after the MAC prior to encryption.
- The final step of SSL Record Protocol processing is to prepare a header consisting of the following fields:

- Content Type (8 bits): The higher-layer protocol used to process the enclosed fragment.
- Major Version (8 bits): Indicates major version of SSL in use. For SSLv3, the value is 3.
- Minor Version (8 bits): Indicates minor version in use. For SSLv3, the value is 0.
- Compressed Length (16 bits): The length in bytes of the plaintext fragment (or compressed fragment if compression is used).The maximum value is 2^{14}+2048.

SSL Record Format



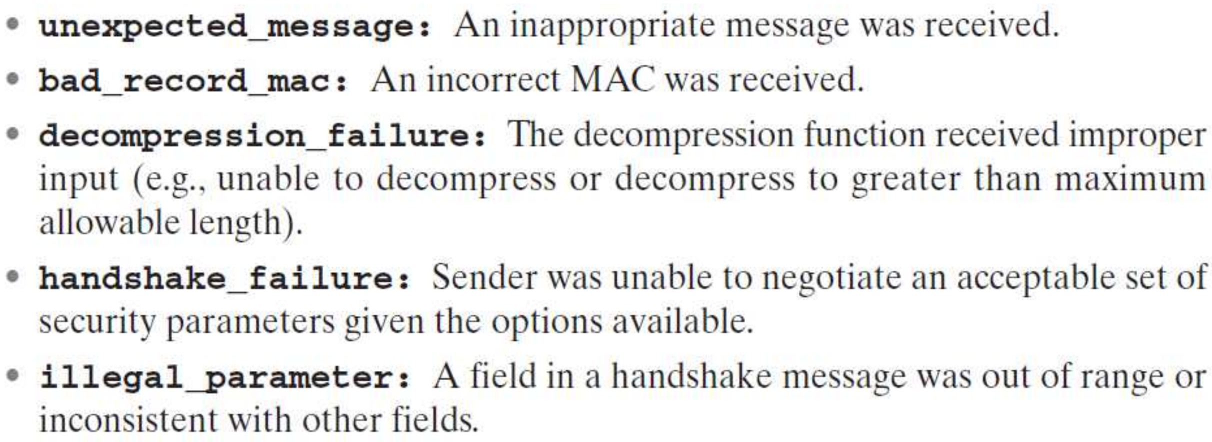
Change Cipher Spec Protocol:

- This protocol consists of a single message, which consists of a single byte with the value 1.
- The sole purpose of this message is to cause the pending state to be copied into the current state, which updates the cipher suite to be used on this connection.

Alert Protocol:

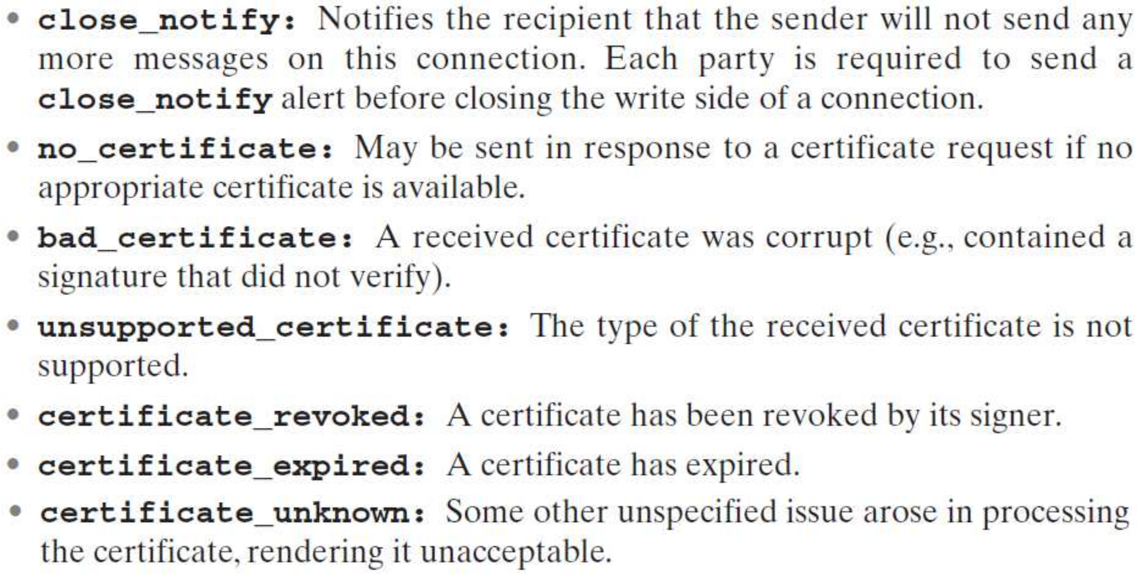
- is used to convey SSL-related alerts to the peer entity.
- Each message in this protocol consists of two bytes .
- The first byte takes the value warning (1) or fatal (2) to convey the severity of the message.
- If the level is fatal, SSL immediately terminates the connection.
- Other connections on the same session may continue, but no new connections on this session may be established.
- The second byte contains a code that indicates the specific alert.

The following alerts that are always fatal.



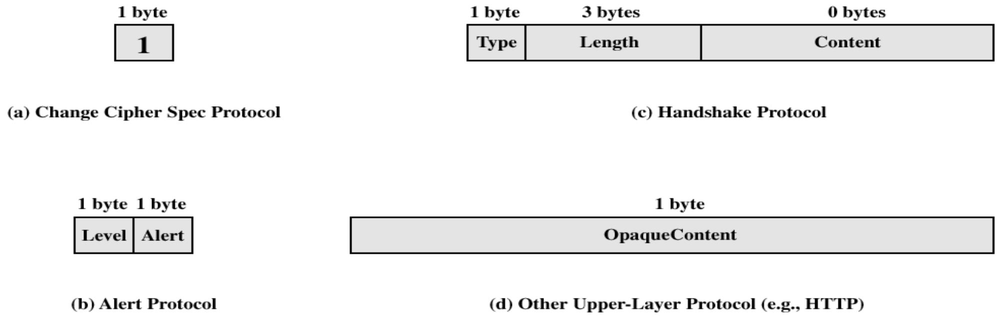
- **unexpected_message:** An inappropriate message was received.
- **bad_record_mac:** An incorrect MAC was received.
- **decompression_failure:** The decompression function received improper input (e.g., unable to decompress or decompress to greater than maximum allowable length).
- **handshake_failure:** Sender was unable to negotiate an acceptable set of security parameters given the options available.
- **illegal_parameter:** A field in a handshake message was out of range or inconsistent with other fields.

Remaining alerts:



- **close_notify:** Notifies the recipient that the sender will not send any more messages on this connection. Each party is required to send a **close_notify** alert before closing the write side of a connection.
- **no_certificate:** May be sent in response to a certificate request if no appropriate certificate is available.
- **bad_certificate:** A received certificate was corrupt (e.g., contained a signature that did not verify).
- **unsupported_certificate:** The type of the received certificate is not supported.
- **certificate_revoked:** A certificate has been revoked by its signer.
- **certificate_expired:** A certificate has expired.
- **certificate_unknown:** Some other unspecified issue arose in processing the certificate, rendering it unacceptable.

Formats of various protocols:



(a) Change Cipher Spec Protocol

(c) Handshake Protocol

(b) Alert Protocol

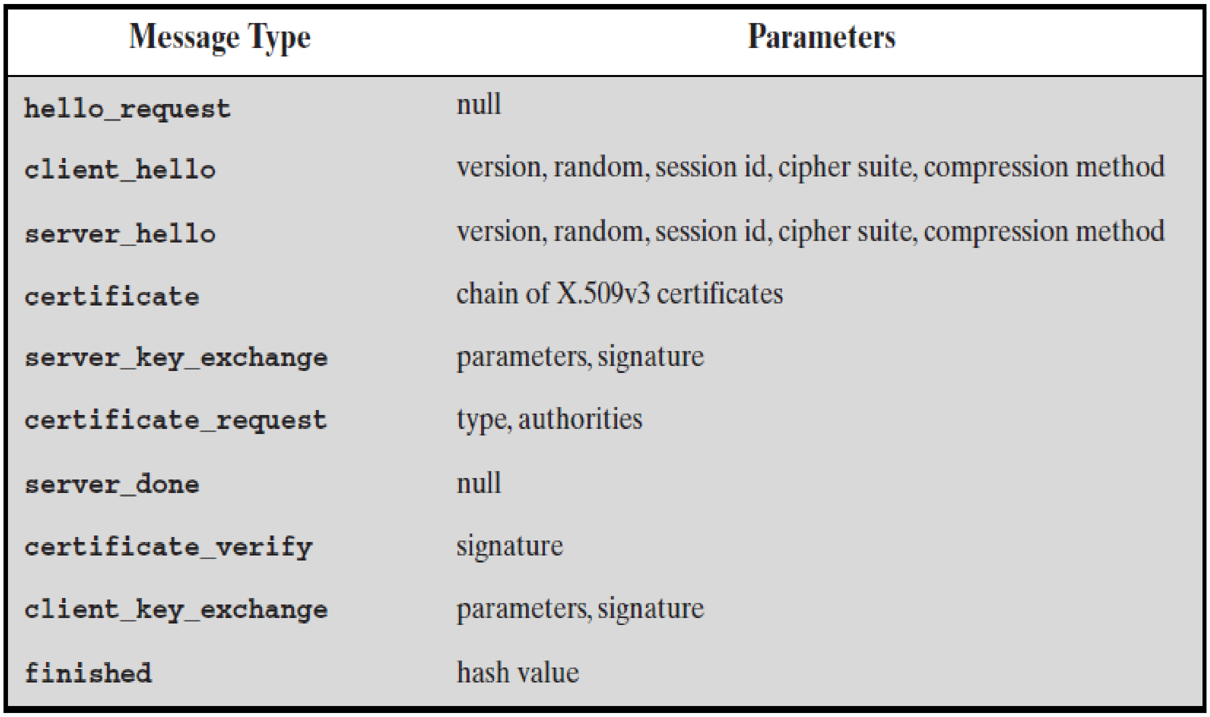
(d) Other Upper-Layer Protocol (e.g., HTTP)

Handshake Protocol:

- The most complex part of SSL.
- Allows the server and client to authenticate each other.
- Negotiate encryption, MAC algorithm and cryptographic keys.
- Used before any application data are transmitted.

- The Handshake Protocol consists of a series of messages exchanged by client and server.
- Each message has three fields:
- Type (1 byte): Indicates one of 10 messages.
- Length (3 bytes): The length of the message in bytes.
- Content (≥bytes): The parameters associated with this message;

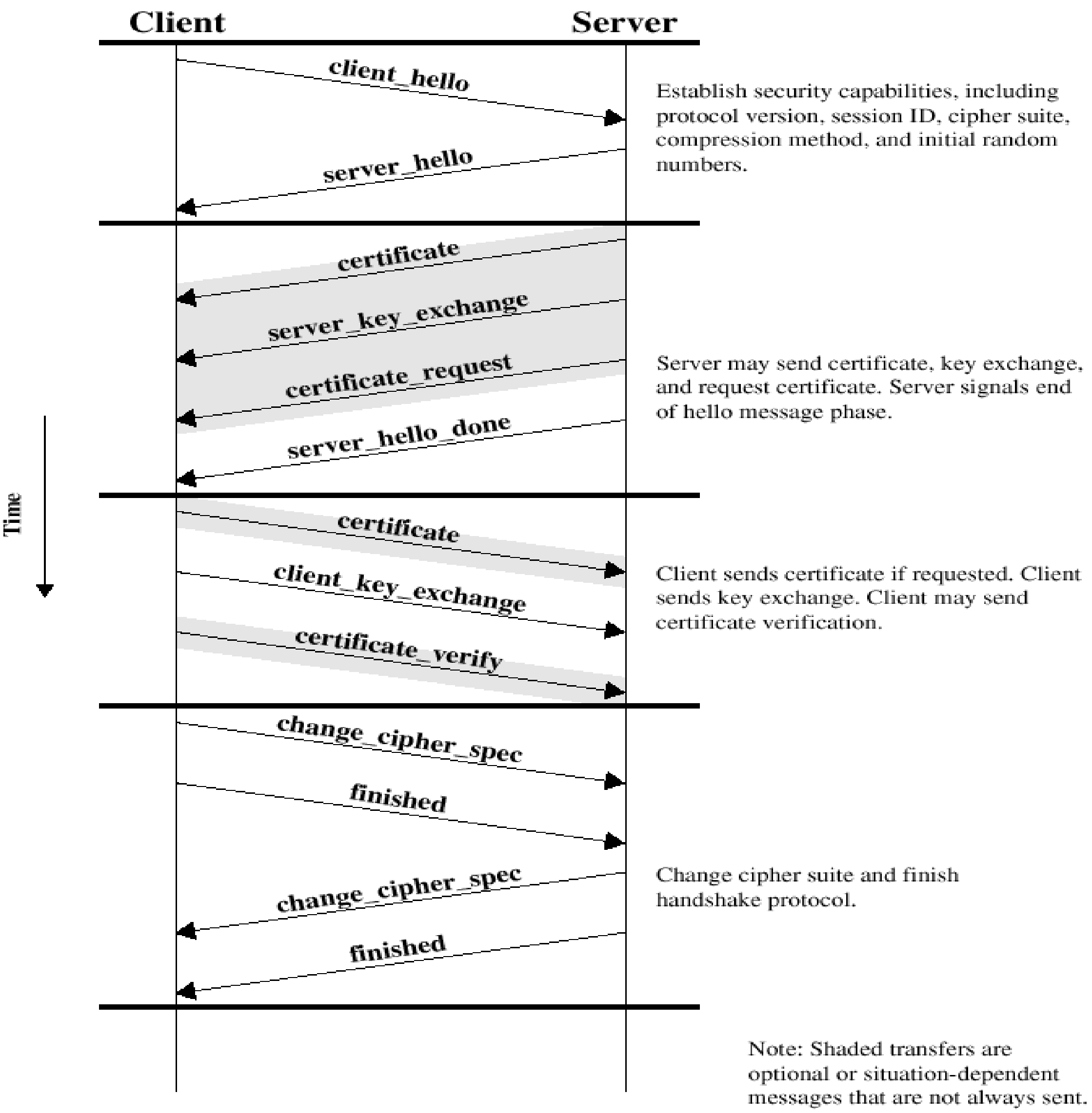
Table 5.2 SSL Handshake Protocol Message Types



Message Type	Parameters
hello_request	null
client_hello	version, random, session id, cipher suite, compression method
server_hello	version, random, session id, cipher suite, compression method
certificate	chain of X.509v3 certificates
server_key_exchange	parameters, signature
certificate_request	type, authorities
server_done	null
certificate_verify	signature
client_key_exchange	parameters, signature
finished	hash value

Figure below shows the initial exchange needed to establish a logical connection between client and server. The exchange can be viewed as having four phases.
PHASE 1. ESTABLISH SECURITY CAPABILITIES
PHASE 2. SERVER AUTHENTICATION AND KEY EXCHANGE
PHASE 3. CLIENT AUTHENTICATION AND KEY EXCHANGE
PHASE 4. FINISH

Handshake Protocol Action:


Client
Server
client_hello
server_hello
Establish security capabilities, including protocol version, session ID, cipher suite, compression method, and initial random numbers.
certificate
server_key_exchange
certificate_request
server_hello_done
Server may send certificate, key exchange, and request certificate. Server signals end of hello message phase.
Time
certificate
client_key_exchange
certificate_verify
Client sends certificate if requested. Client sends key exchange. Client may send certificate verification.
change_cipher_spec
finished
change_cipher_spec
finished
Change cipher suite and finish handshake protocol.
Note: Shaded transfers are optional or situation-dependent messages that are not always sent.

- *PHASE 1. ESTABLISH SECURITY CAPABILITIES*
    - This phase is used to initiate a logical connection and to establish the security capabilities that will be associated with it.
    - The exchange is initiated by the client, which sends a client_hello message with the following parameters:
    - Version: The highest SSL version understood by the client.
    - Random: A client-generated random structure consisting of a 32-bit timestamp and 28 bytes generated by a secure random number generator.
    - These values serve as nonces and are used during key exchange to prevent replay attacks.
- Session ID:
    - A variable-length session identifier.
    - A nonzero value indicates that the client wishes to update the parameters of an existing connection or to create a new connection on this session.

  - A zero value indicates that the client wishes to establish a new connection on a new session.
- CipherSuite:
  - This is a list that contains the combinations of cryptographic algorithms supported by the client, in decreasing order of preference.
  - Each element of the list (each cipher suite) defines both a key exchange algorithm and a CipherSpec;
  - Compression Method: This is a list of the compression methods the client supports.
- After sending the *client_hello* message, the client waits for the *server_hello* message, which contains the same parameters as the *client_hello* message.
- For the *server_hello* message, the following conventions apply.
- The Version field contains the lower of the versions suggested by the client and the highest supported by the server.
- The Random field is generated by the server and is independent of the client's Random field.
- If the *SessionID* field of the client was nonzero, the same value is used by the server; otherwise the server's *SessionID* field contains the value for a new session.
- The *CipherSuite* field contains the single cipher suite selected by the server from those proposed by the client.
- The Compression field contains the compression method selected by the server from those proposed by the client.
- The first element of the *CipherSuite* parameter is the key exchange method (i.e., the means by which the cryptographic keys for conventional encryption and MAC are exchanged).The following key exchange methods are supported.
  - RSA: The secret key is encrypted with the receiver's RSA public key.
  - A publickey certificate for the receiver's key must be made available.
  - ***Fixed Diffie-Hellman***: This is a Diffie-Hellman key exchange in which the server's certificate contains the Diffie-Hellman public parameters signed by the certificate authority (CA).
  - The client provides its Diffie-Hellman public-key parameters either in a certificate, if client authentication is required, or in a key exchange message.
  - This method results in a fixed secret key between two peers based on the Diffie-Hellman calculation using the fixed public keys.
  - ***Ephemeral Diffie-Hellman:*** This technique is used to create ephemeral (temporary, one-time) secret keys. In this case, the Diffie-Hellman public keys are exchanged, signed using the sender's private RSA or DSS key.
  - The receiver can use the corresponding public key to verify the signature.
  - Certificates are used to authenticate the public keys. This would appear to be the most secure of the three Diffie-Hellman options, because it results in a temporary, authenticated key.
  - **Anonymous Diffie-Hellman:** The base Diffie-Hellman algorithm is used with no authentication.
  - That is, each side sends its public Diffie-Hellman parameters to the other with no authentication.
  - This approach is vulnerable to man-in-the middle attacks, in which the attacker conducts anonymous Diffie-Hellman with both parties.
  - **Fortezza:** the technique defined by US National security Agency.
  - Developed for defense department.
- Following the definition of a key exchange method is the CipherSpec, which includes the following fields.

- Cipher Algorithm: Any of the algorithms mentioned earlier: RC4, RC2, DES, 3DES, DES40, IDEA, or Fortezza.
- MACAlgorithm: MD5 or SHA-1
- CipherType: Stream or Block
- IsExportable: True or False
- HashSize: 0, 16 (for MD5), or 20 (for SHA-1) bytes
- Key Material: A sequence of bytes that contain data used in generating the write keys
- IV Size: The size of the Initialization Value for Cipher Block Chaining (CBC) encryption

SSL Handshake Protocol / Phase 2

Server certificate and key exchange messages:

- certificate
  - required for every key exchange method except for anonymous DH
  - contains one or a chain of X.509 certificates (up to a known root CA)
  - may contain
    - public RSA key suitable for encryption, or
    - public RSA or DSS key suitable for signing only, or
    - fix DH parameters
- server_key_exchange
  - sent only if the certificate does not contain enough information to complete the key exchange (e.g., the certificate contains an RSA signing key only)
  - may contain
    - public RSA key (exponent and modulus), or
    - DH parameters (p, g, public DH value), or
    - Fortezza parameters
  - digitally signed
    - if DSS: SHA-1 hash of (client_random | server_random | server_params) is signed
    - if RSA: MD5 hash and SHA-1 hash of (client_random | server_random | server_params) are concatenated and encrypted with the private RSA key

Certificate request and server hello done msgs:

- certificate_request
  - sent if the client needs to authenticate itself
  - specifies which type of certificate is requested (rsa_sign, dss_sign, rsa_fixed_dh, dss_fixed_dh, …)
- server_hello_done
  - sent to indicate that the server is finished its part of the key exchange
  - after sending this message the server waits for client response
  - the client should verify that the server provided a valid certificate and the server parameters are acceptable

SSL Handshake Protocol / Phase 3
Client authentication and key exchange

- certificate
  - sent only if requested by the server
  - may contain
    - public RSA or DSS key suitable for signing only, or
    - fix DH parameters
- client_key_exchange

    - always sent (but it is empty if the key exchange method is fix DH)
    - may contain
        - RSA encrypted pre-master secret, or
        - client one-time public DH value, or
        - Fortezza key exchange parameters
- certificate_verify
    - sent only if the client sent a certificate
    - provides client authentication
    - contains signed hash of all the previous handshake messages
        - if DSS: SHA-1 hash is signed
        - if RSA: MD5 and SHA-1 hash is concatenated and encrypted with the private key

MD5(master_secret|pad_2|MD5(handshake_messages | master_secret | pad_1))
SHA(master_secret|pad_2|SHA(handshake_messages | master_secret | pad_1))

SSL Handshake Protocol / Phase 4:
Finished messages:

- finished
    - sent immediately after the change_cipher_spec message
    - first message that uses the newly negotiated algorithms, keys, IVs, etc.
    - used to verify that the key exchange and authentication was successful
    - contains the MD5 and SHA-1 hash of all the previous handshake messages:

MD5(master_secret|pad_2|MD5(handshake_messages|sender|master_secret| pad_1)) |SHA(master_secret|pad_2|SHA(handshake_messages|sender| master_secret| pad_1))

where "sender" is a code that identifies that the sender is the client or the server (client: 0x434C4E54; server: 0x53525652)

Cryptographic Computations:
Two further items are of interest: (1) the creation of a shared master secret by means of the key exchange and (2) the generation of cryptographic parameters from the master secret.

MASTER SECRET CREATION The shared master secret is a one-time 48-byte value (384 bits) generated for this session by means of secure key exchange. The creation is in two stages. First, a *pre_master_secret* is exchanged. Second, the *master_secret* is calculated by both parties. For *pre_master_secret* exchange, there are two possibilities.

- **RSA:** A 48-byte *pre_master_secret* is generated by the client, encrypted with the server's public RSA key, and sent to the server. The server decrypts the ciphertext using its private key to recover the *pre_master_secret*.
- **Diffie-Hellman:** Both client and server generate a Diffie-Hellman public key. After these are exchanged, each side performs the Diffie-Hellman calculation to create the shared pre_master_secret.

Both sides now compute the master_secret as

master_secret=MD5(pre_master_secret||SHA('A'||pre_master_secret|| ClientHello.random ||ServerHello.random))||MD5(pre_master_secret||SHA('BB'||pre_master_secret|| ClientHello.random ||ServerHello.random))||MD5(pre_master_secret || SHA('CCC' || pre_master_secret || ClientHello.random ||ServerHello.random))

where ClientHello.random and ServerHello.random are the two nonce values exchanged in the initial hello messages.

GENERATION OF CRYPTOGRAPHIC PARAMETERS CipherSpecs require a client write MAC secret, a server write MAC secret, a client write key, a server write key, a client write IV, and a server write IV, which are generated from the master secret in that order. These parameters are generated from the master secret by hashing the master secret into a sequence of secure bytes of sufficient length for all needed parameters. The generation of the key material from the master secret uses the same format for generation of the master secret from the pre-master secret as

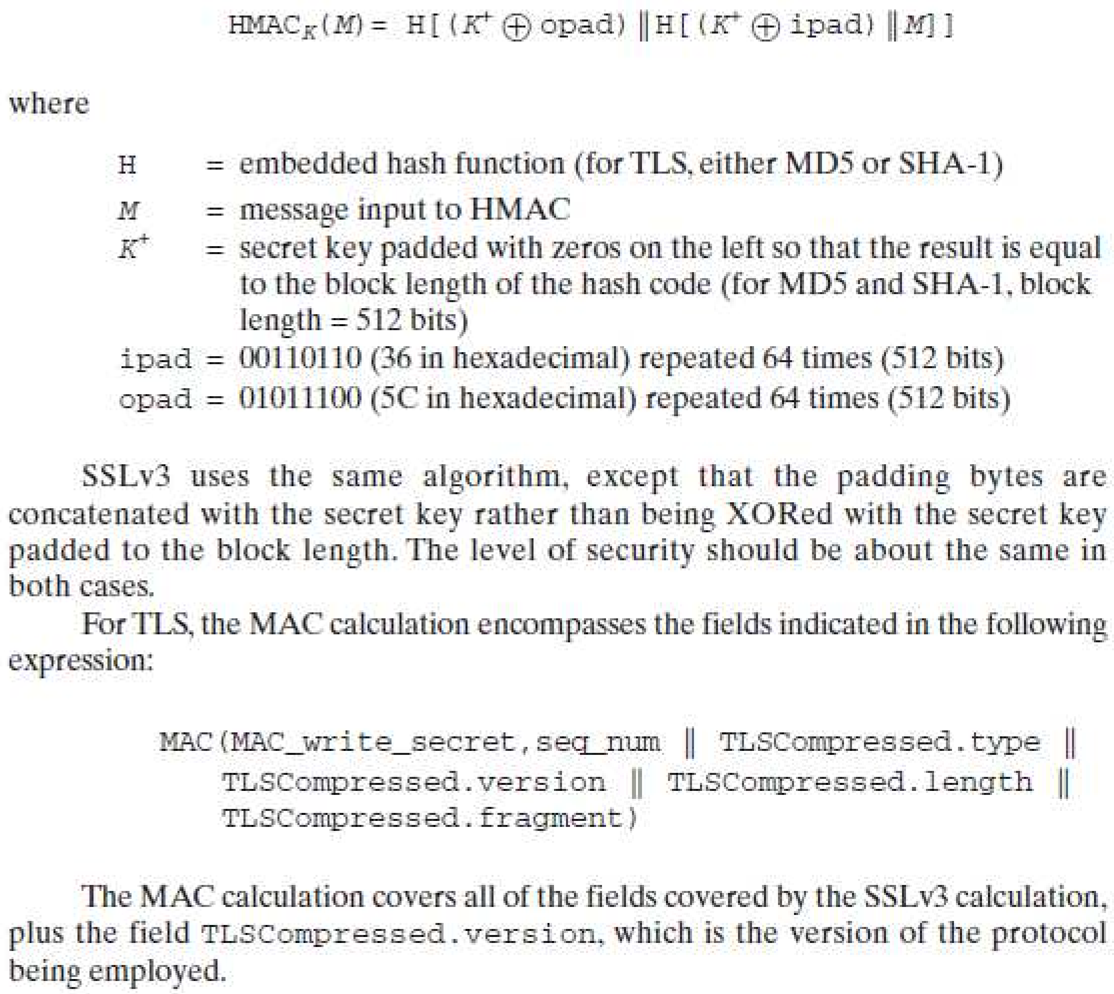
key_block = MD5(master_secret| SHA('A'||master_secret||ServerHello.random|| ClientHello.random)) ||MD5(master_secret||SHA('BB'||master_secret||ServerHello.random || ClientHello.random)) ||MD5(master_secret || SHA('CCC' || master_secret || ServerHello.random || ClientHello.random)) ||...

until enough output has been generated. The result of this algorithmic structure is a pseudorandom function.We can view the master_secret as the pseudorandom seed value to the function. The client and server random numbers can be viewed as salt values to complicate cryptanalysis (see Chapter 9 for a discussion of the use of salt values).

Transport Layer Security:

TLS is an IETF standardization initiative whose goal is to produce an Internet standard version of SSL. TLS is defined as a Proposed Internet Standard in RFC 5246. RFC 5246 is very similar to SSLv3. In this section, we highlight the differences.

- Differences in the:
  - version number
  - message authentication code
  - pseudorandom function
  - alert codes
  - cipher suites
  - client certificate types
  - certificate_verify and finished message
  - cryptographic computations
  - padding
- **Version Number**
  - The TLS Record Format is the same as that of the SSL Record Format (Figure 5.4), and the fields in the header have the same meanings. The one difference is in version values. For the current version of TLS, the major version is 3 and the minor version is 3.
- **Message Authentication Code**
  - There are two differences between the SSLv3 and TLS MAC schemes: the actual algorithm and the scope of the MAC calculation. TLS makes use of the HMAC algorithm defined in RFC 2104. Recall from Chapter 3 that HMAC is defined as



$$\mathrm{HMAC}_K(M) = \mathrm{H}[(K^+ \oplus \mathrm{opad}) \| \mathrm{H}[(K^+ \oplus \mathrm{ipad}) \| M]]$$

where

H = embedded hash function (for TLS, either MD5 or SHA-1)

M = message input to HMAC

K^+ = secret key padded with zeros on the left so that the result is equal to the block length of the hash code (for MD5 and SHA-1, block length = 512 bits)

ipad = 00110110 (36 in hexadecimal) repeated 64 times (512 bits)

opad = 01011100 (5C in hexadecimal) repeated 64 times (512 bits)

SSLv3 uses the same algorithm, except that the padding bytes are concatenated with the secret key rather than being XORed with the secret key padded to the block length. The level of security should be about the same in both cases.

For TLS, the MAC calculation encompasses the fields indicated in the following expression:

```
MAC(MAC_write_secret,seq_num || TLSCompressed.type ||
    TLSCompressed.version || TLSCompressed.length ||
    TLSCompressed.fragment)
```

The MAC calculation covers all of the fields covered by the SSLv3 calculation, plus the field TLSCompressed.version, which is the version of the protocol being employed.

Pseudorandom Function

TLS makes use of a pseudorandom function referred to as PRF to expand secrets into blocks of data for purposes of key generation or validation.The objective is to make use of a relatively small shared secret value but to generate longer blocks of data in a way that is secure from the kinds of attacks made on hash functions and MACs.The PRF is based on the data expansion function (Figure below) given as

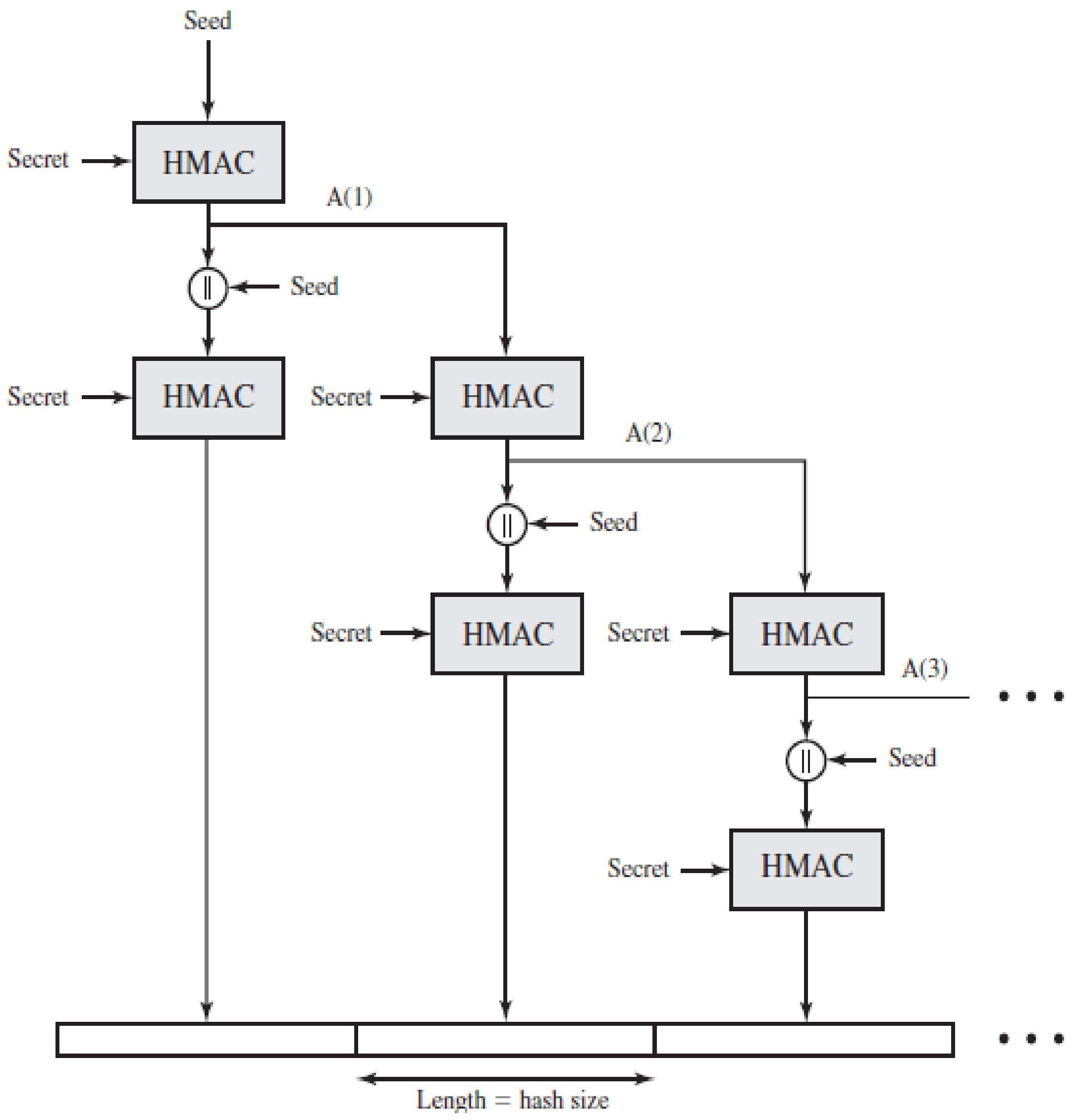
P_hash(secret, seed)= HMAC_hash(secret,A(1) || seed) || HMAC_hash(secret, A(2) || seed) || HMAC_hash(secret, A(3) || seed) || . . .

where A() is defined as

A(0) = seed

A(i) = HMAC_hash(secret,A(*i* – 1))

PRF(secret, label, seed) =P_MD5(secret_left, label | seed) Å P_SHA(secret_right, label | seed)



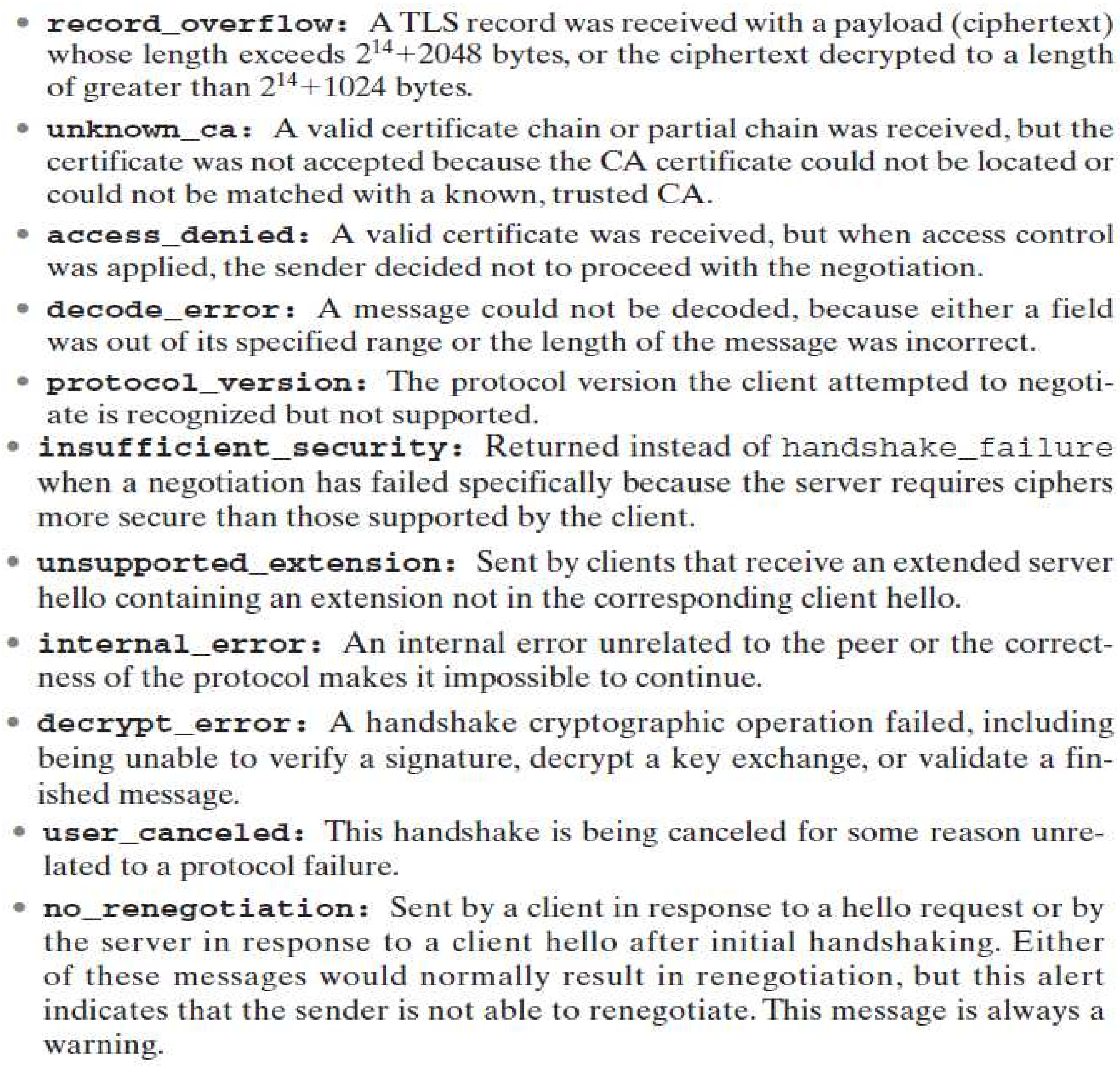
To make PRF as secure as possible, it uses two hash algorithms in a way that should guarantee its security if either algorithm remains secure. PRF is defined as

PRF(secret, label, seed) = P_hash(S1,label || seed)

PRF takes as input a secret value, an identifying label, and a seed value and produces an output of arbitrary length.

Alert Codes:

- TLS supports all of the alert codes defined in SSLv3 with the exception of no_certificate.
- A number of additional codes are defined in TLS; of these, the following are always fatal.



- **record_overflow:** A TLS record was received with a payload (ciphertext) whose length exceeds 2^{14}+2048 bytes, or the ciphertext decrypted to a length of greater than 2^{14}+1024 bytes.
- **unknown_ca:** A valid certificate chain or partial chain was received, but the certificate was not accepted because the CA certificate could not be located or could not be matched with a known, trusted CA.
- **access_denied:** A valid certificate was received, but when access control was applied, the sender decided not to proceed with the negotiation.
- **decode_error:** A message could not be decoded, because either a field was out of its specified range or the length of the message was incorrect.
- **protocol_version:** The protocol version the client attempted to negotiate is recognized but not supported.
- **insufficient_security:** Returned instead of handshake_failure when a negotiation has failed specifically because the server requires ciphers more secure than those supported by the client.
- **unsupported_extension:** Sent by clients that receive an extended server hello containing an extension not in the corresponding client hello.
- **internal_error:** An internal error unrelated to the peer or the correctness of the protocol makes it impossible to continue.
- **decrypt_error:** A handshake cryptographic operation failed, including being unable to verify a signature, decrypt a key exchange, or validate a finished message.
- **user_canceled:** This handshake is being canceled for some reason unrelated to a protocol failure.
- **no_renegotiation:** Sent by a client in response to a hello request or by the server in response to a client hello after initial handshaking. Either of these messages would normally result in renegotiation, but this alert indicates that the sender is not able to renegotiate. This message is always a warning.

Secure Electronic Transactions:

- An open encryption and security specification.
- Protect credit card transaction on the Internet.
- Companies involved:
  - MasterCard, Visa, IBM, Microsoft, Netscape, RSA, Terisa and Verisign
- Not a payment system.
- Set of security protocols and formats.

SET Services:

- Provides a secure communication channel in a transaction.
- Provides tust by the use of X.509v3 digital certificates.
- Ensures privacy.

SET Overview:

A good way to begin our discussion of SET is to look at the business requirements for SET, its key features, and the participants in SET transactions.

Requirements

Book 1 of the SET specification lists the following business requirements for secure payment processing with credit cards over the Internet and other networks:

Provide confidentiality of payment and ordering information: It is necessary to assure cardholders that this information is safe and accessible only to the intended recipient. Confidentiality also reduces the risk of fraud by either party to the transaction or by malicious third parties. SET uses encryption to provide confidentiality.

Ensure the integrity of all transmitted data: That is, ensure that no changes in content occur during transmission of SET messages. Digital signatures are used to provide integrity.

Provide authentication that a cardholder is a legitimate user of a credit card account: A mechanism that links a cardholder to a specific account number reduces the incidence of fraud and the overall cost of payment processing. Digital signatures and certificates are used to verify that a cardholder is a legitimate user of a valid account.

Provide authentication that a merchant can accept credit card transactions through its relationship with a financial institution: This is the complement to the preceding requirement. Cardholders need to be able to identify merchants with whom they can conduct secure transactions. Again, digital signatures and certificates are used.

Ensure the use of the best security practices and system design techniques to protect all legitimate parties in an electronic commerce transaction: SET is a well-tested specification based on highly secure cryptographic algorithms and protocols.

Create a protocol that neither depends on transport security mechanisms nor prevents their use: SET can securely operate over a "raw" TCP/IP stack. However, SET does not interfere with the use of other security mechanisms. Such as IPSec and SSL/TLS.

Facilitate and encourage interoperability among software and network providers: The SET protocols and formats are independent of hardware platform, operating system, and Web software.

Key Features of SET:

- Confidentiality of information
- Integrity of data
- Cardholder account authentication
- Merchant authentication

To meet the requirements just outlined, SET incorporates the following features:

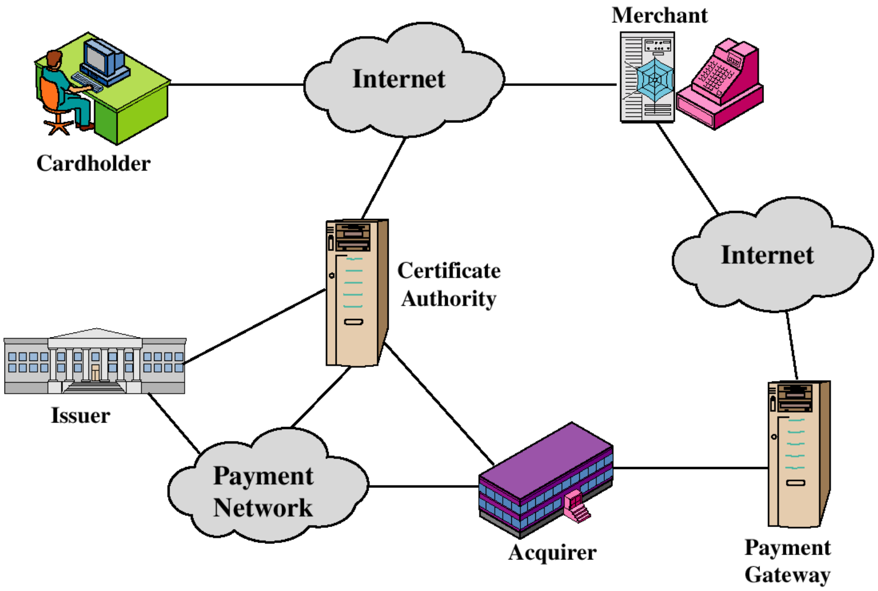
Confidentiality of information: Cardholder account and payment information is secured as it travels across the network. An interesting and important feature of SET is that it prevents the merchant from learning the cardholder's credit card number; this is only provided to the issuing bank. Conventional encryption by DES is used to provide confidentiality.

Integrity of data: Payment information sent from cardholders to merchants includes order information, personal data, and payment instructions. SET guarantees that these message contents are not altered in transit. RSA digital signatures, using SHA-1 hash codes, provide message integrity. Certain messages are also protected by HMAC using SHA-1.

Cardholder account authentication: SET enables merchants to verify that a cardholder is a legitimate user of a valid card account number. SET uses X.509~3 digital certificates with RSA signatures for this purpose.

Merchant authentication: SET enables cardholders to verify that a merchant has a relationship with a financial institution allowing it to accept payment cards. SET uses X.509~3 digital certificates with RSA signatures for this purpose.

SET Participants:



Cardholder: In the electronic environment, consumers and corporate purchasers interact with merchants from personal computers over the Internet. A cardholder is an authorized holder of a payment card (e.g., Mastercard, Visa) that has been issued by an issuer.

Merchant: A merchant is a person or organization that has goods or services to sell to the cardholder. Typically, these goods and services are offered via a Web site or by electronic mail. A merchant that accepts payment cards must have a relationship with an acquirer.

Issuer: This is a financial institution, such as a bank, that provides the cardholder with the payment card. Typically, accounts are applied for and opened by mail or in person. Ultimately, it is the issuer that is responsible for the payment of the debt of the cardholder.

Acquirer: This is a financial institution that establishes an account with a merchant and processes payment card authorizations and payments. Merchants will usually accept more than one credit card brand but do not want to deal with multiple bankcard associations or with multiple individual issuers. The acquirer provides authorization to the merchant that a given card account is active and that the proposed purchase does not exceed the credit limit. The acquirer also provides electronic transfer of payments to the merchant's account. Subsequently, the acquirer is reimbursed by the issuer over some sort of payment network for electronic funds transfer.

Payment Gateway: This is a function operated by the acquirer or a designated third party that processes merchant payment messages. The payment gateway interfaces between SET and the existing bankcard payment networks for authorization and payment functions. The merchant exchanges SET messages with the payment gateway over the Internet, while the payment gateway has some direct or network connection to the acquirer's financial processing system.

Certification Authority (CA): This is an entity that is trusted to issue X.509~3 public-key certificates for cardholders, merchants, and payment gateways. The success of SET will depend on the existence of a CA infrastructure available for this purpose. As was discussed in previous chapters, a hierarchy of CAs is used, so that participants need not be directly certified by a root authority.

We now briefly describe the sequence of events that are required for a transaction. We will then look at some of the cryptographic details.

1. The customer opens an account. The customer obtains a credit card account, such as Mastercard or Visa, with a bank that supports electronic payment and SET.

2. The customer receives a certificate. After suitable verification of identity, the customer receives an X.509~3 digital certificate, which is signed by the bank. The certificate verifies the customer's RSA public key and its expiration date. It also establishes a relationship, guaranteed by the bank, between the customer's key pair and his or her credit card.

3. Merchants have their own certificates. A merchant who accepts a certain brand of card must be in possession of two certificates for two public keys owned by the merchant: one for signing messages, and one for key exchange. The merchant also needs a copy of the payment gateway's public-key certificate.

4. The customer places an order. This is a process that may involve the customer first browsing through the merchant's Web site to select items and determine the price. The customer then sends a list of the items to be purchased to the merchant, who returns an order form containing the list of items, their price, a total price, and an order number.

5. The merchant is verified. In addition to the order form, the merchant sends a copy of its certificate, so that the customer can verify that he or she is dealing with a valid store.

6. The order and payment are sent. The customer sends both order and payment information to the merchant, along with the customer's certificate. The order confirms the purchase of the items in the order form. The payment contains credit card details. The payment information is encrypted in such a way that it cannot be read by the merchant. The customer's certificate enables the merchant to verify the customer.

7. The merchant requests payment authorization. The merchant sends the payment information to the payment gateway, requesting authorization that the customer's available credit is sufficient for this purchase.

8. The merchant confirms the order. The merchant sends confirmation of the order to the customer.

9. The merchant provides the goods or service. The merchant ships the goods or provides the service to the customer.

10. The merchant requests payment. This request is sent to the payment gateway, which handles all of the payment processing.

Dual Signature:

Before looking at the details of the SET protocol, let us discuss an important innovation introduced in SET: the dual signature. The purpose of the dual signature is to link two messages that are intended for two different recipients. In this case, the customer want to send the order information (OI) to the merchant and the payment information (PI) to the bank. The merchant does not need to know the customer's credit card number, and the bank does not need to know the details of the customer's order. The customer is afforded extra protection in terms of privacy by keeping these two items separate. However, the two items must be linked in a way that can be used to resolve disputes if necessary. The link is needed so that the customer can prove that this payment is intended for this order and not for some other goods or service.

To see the need for the link, suppose that the customer sends the merchant two messages-a signed OI and a signed PI-and the merchant passes the PI on to the bank. If the merchant can capture another OI from this customer, the merchant could claim that this OI goes with the PI rather than the original OI. The linkage prevents this.

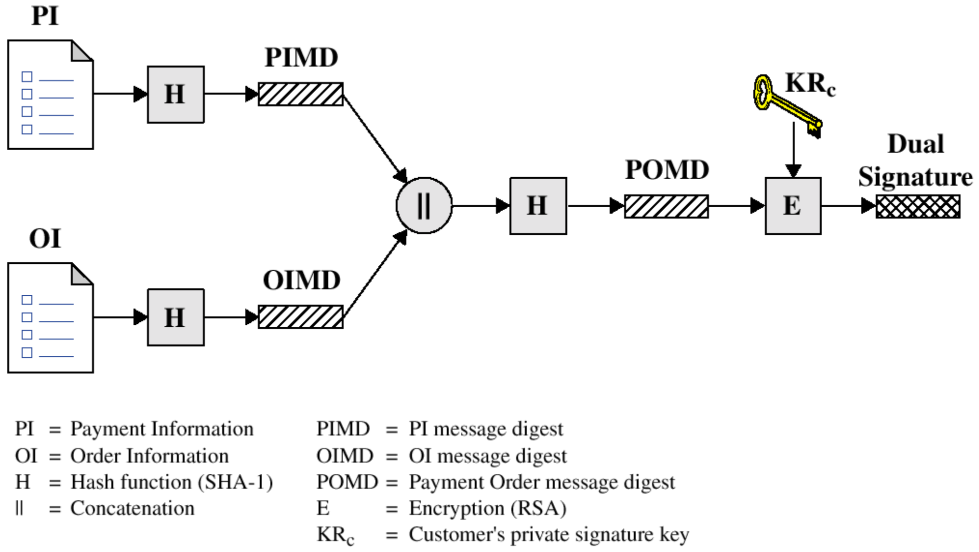


Figure above shows the use of a dual signature to meet the requirement of the preceding paragraph. The customer takes the hash (using SHA-1) of the PI and the hash of the OI. These two hashes are then concatenated and the hash of the result is taken. Finally, the customer encrypts the final hash with his or her private signature key, creating the dual signature. The operation can be summarized as follows:

$$DS = E_{KR}c[H(H(PI)||H(OI))]$$

where KR, is the customer's private signature key. Now suppose that the merchant is in possession of the dual signature (DS), the OI, and the message digest for the PI (PIMD). The merchant also has the public key of the customer, taken from the customer's certificate. Then the merchant can compute the following two quantities:

$$H(PIMD||\ (H(OI))\text{ and }D_{KU}c[DS]$$

where KU, is the customer's public signature key. If these two quantities are equal. then the merchant has verified the signature. Similarly, if the bank is in possession of DS, PI, the message digest for OI (OIMD), and the customer's public key, then
the bank can compute the following:

$$H(H(PI)||OIMD)\text{ and }D_{KUc}[DS]$$

Again, if these two quantities are equal, then the bank has verified the signature.
In summary,

1. The merchant has received OI and verified the signature.
2. The bank has received PI and verified the signature.
3. The customer has linked the OI and PI and can prove the linkage.

Purchase Request

Before the Purchase Request exchange begins, the cardholder has completed browsing, selecting, and ordering. The end of this preliminary phase occurs when the merchant sends a completed order form to the customer. All of the preceding occurs without the use of SET.

The purchase request exchange consists of four messages: Initiate Request, Initiate Response, Purchase Request, and Purchase Response.

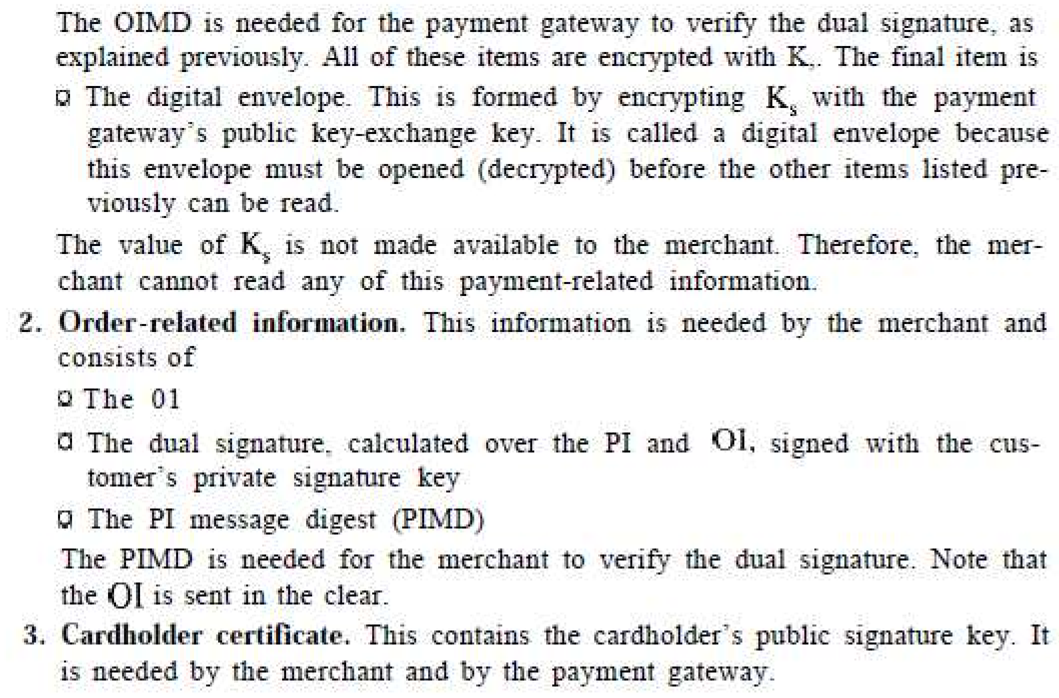
In order to send SET messages to the merchant, the cardholder must have a copy of the certificates of the merchant and the payment gateway. The customer requests the certificates in the **Initiate Request** message, sent to the merchant. This message includes the brand of the credit card that the customer is using. The message also includes an ID assigned to this request/response pair by the customer and a nonce used to ensure timeliness.

The merchant generates a response and signs it with its private signature key. The response includes the nonce from the customer, another nonce for the customer to return in the next message, and a transaction ID for this purchase transaction. In addition to the signed response, the **Initiate Response** message includes the merchant's signature certificate and the payment gateway's key exchange certificate.

The cardholder verifies the merchant and gateway certificates by means of their respective CA signatures and then creates the OI and PI. The transaction ID assigned by the merchant is placed in both the 01 and PI. The OI does not contain explicit order data such as the number and price of items. Rather, it contains an order reference generated in the exchange between merchant and customer during the shopping phase before the first SET message. Next, the cardholder prepares the **Purchase Request** message (Figure 14.10). For this purpose, the cardholder generates a one-time symmetric encryption key, K,. The message includes the following:

1. Purchase-related information. This information will be forwarded to the payment gateway by the merchant and consists of

- a The PI
- The dual signature, calculated over the PI and 01, signed with the customer's
- private signature key
- The OI message digest (OIMD)



The OIMD is needed for the payment gateway to verify the dual signature, as explained previously. All of these items are encrypted with K,. The final item is

□ The digital envelope. This is formed by encrypting K_s with the payment gateway's public key-exchange key. It is called a digital envelope because this envelope must be opened (decrypted) before the other items listed previously can be read.

The value of K_s is not made available to the merchant. Therefore, the merchant cannot read any of this payment-related information.

2. **Order-related information.** This information is needed by the merchant and consists of

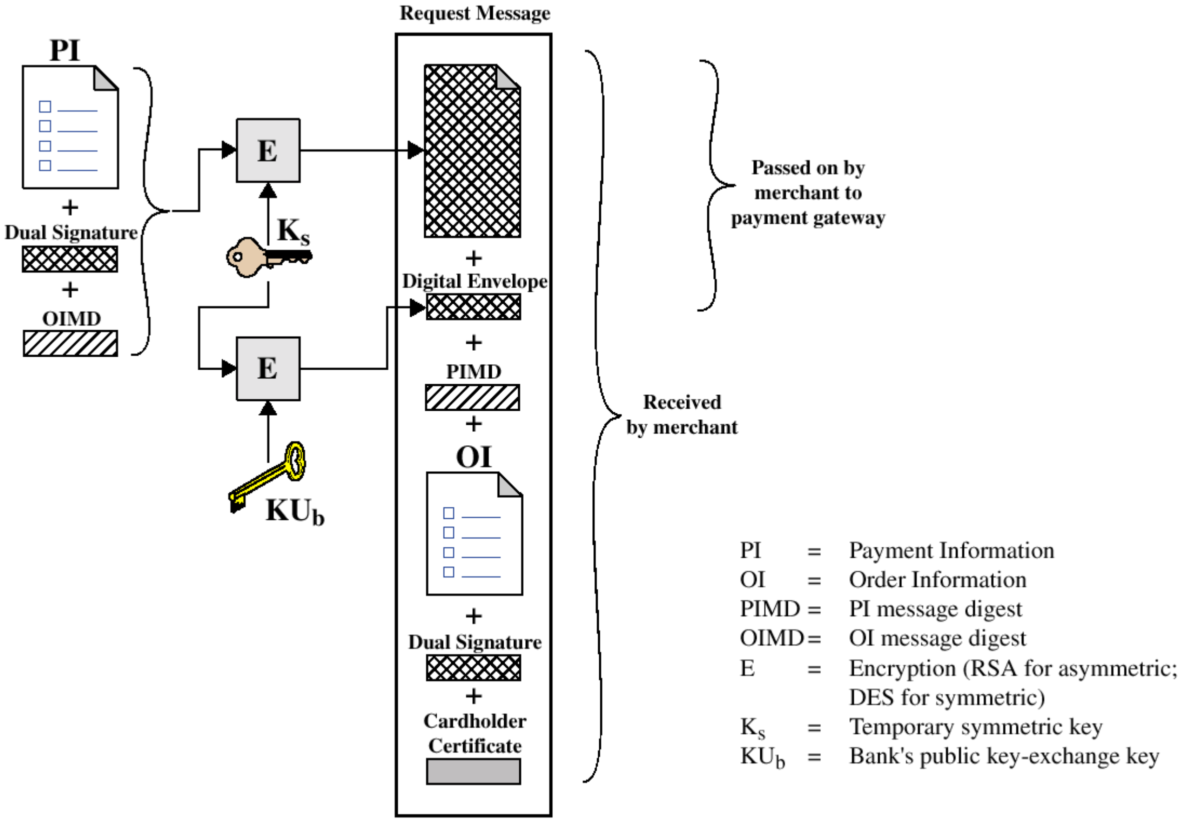
□ The 01

□ The dual signature, calculated over the PI and OI, signed with the customer's private signature key

□ The PI message digest (PIMD)

The PIMD is needed for the merchant to verify the dual signature. Note that the OI is sent in the clear.

3. **Cardholder certificate.** This contains the cardholder's public signature key. It is needed by the merchant and by the payment gateway.



PI = Payment Information
OI = Order Information
PIMD = PI message digest
OIMD = OI message digest
E = Encryption (RSA for asymmetric; DES for symmetric)
K_s = Temporary symmetric key
KU_b = Bank's public key-exchange key

When the merchant receives the Purchase Request message, it performs the following actions:

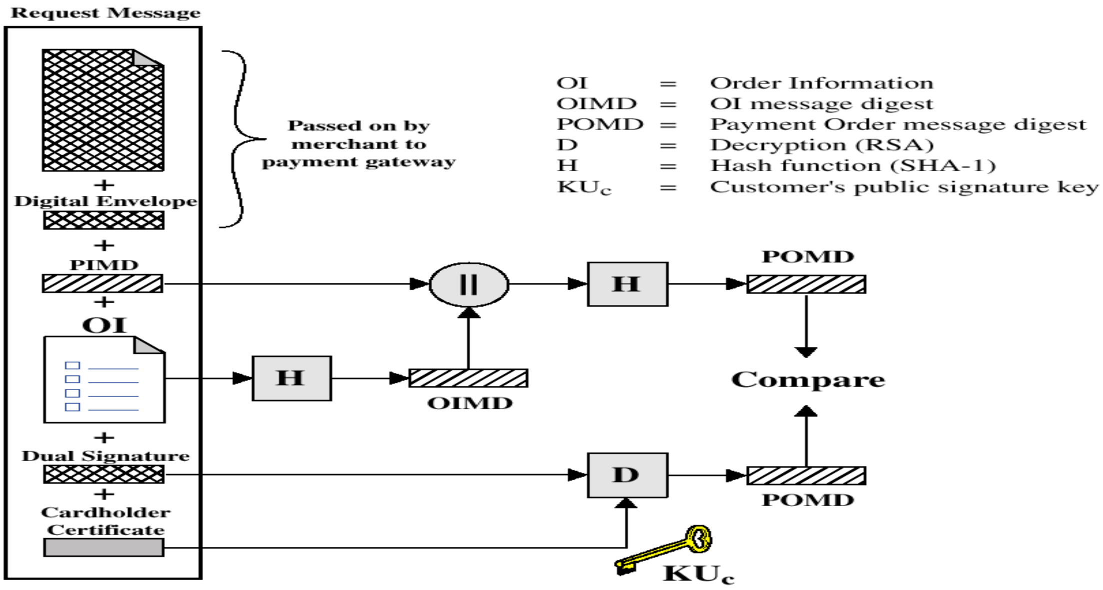
1. Verifies the cardholder certificates by means of its CA signatures.
2. Verifies the dual signature using the customer's public signature key. This ensures that the order has not been tampered with in transit and that it was signed using the cardholder's private signature key.
3. Processes the order and forwards the payment information to the payment gateway for authorization (described later).
4. Sends a purchase response to the cardholder.

The **Purchase Response** message includes a response block that acknowledges the order and references the corresponding transaction number. This block is signed by the merchant using its private signature key. The block and its signature are sent to the customer, along with the merchant's signature certificate.

When the cardholder software receives the purchase response message, it verifies the merchant's certificate and then verifies the signature on the response block. Finally, it takes some action based on the response, such as displaying a message to the user or updating a database with the status of the order.

Payment Authorization

During the processing of an order from a cardholder, the merchant authorizes the transaction with the payment gateway. The payment authorization ensures that the transaction was approved by the issuer. This authorization guarantees that the



merchant will receive payment; the merchant can therefore provide the services or goods to the customer. The payment authorization exchange consists of two messages: Authorization Request and Authorization response.

The merchant sends an **Authorization Request** message to the payment gateway consisting of

1. **Purchase-related information.** This information was obtained from the customer and consists of:
   - □ The PI
   - □ The dual signature, calculated over the PI and OI, signed with the customer's private signature key
   - □ The 01 message digest (OIMD)
   - □ The digital envelope
2. **Authorization-related information.** This information is generated by the merchant and consists of
   - □ An authorization block that includes the transaction ID, signed with the merchant's private signature key and encrypted with a one-time symmetric key generated by the merchant
   - □ A digital envelope. This is formed by encrypting the one-time key with the payment gateway's public key-exchange key.
3. **Certificates.** The merchant includes the cardholder's signature key certificate (used to verify the dual signature), the merchant's signature key certificate (used to verify the merchant's signature), and the merchant's key-exchange certificate (needed in the payment gateway's response).

The payment gateway performs the following tasks:

1. Verifies all certificates
2. Decrypts the digital envelope of the authorization block to obtain the symmetric key and then decrypts the authorization block
3. Verifies the merchant's signature on the authorization block
4. Decrypts the digital envelope of the payment block to obtain the symmetric key and then decrypts the payment block
5. Verifies the dual signature on the payment block
6. Verifies that the transaction ID received from the merchant matches that in the PI received (indirectly) from the customer
7. Requests and receives an authorization from the issuer

Having obtained authorization from the issuer, the payment gateway returns an **Authorization Response** message to the merchant. It includes the following elements:

1. **Authorization-related information.** Includes an authorization block, signed with the gateway's private signature key and encrypted with a one-time symmetric key generated by the gateway. Also includes a digital envelope that contains the one-time key encrypted with the merchants public key-exchange key.
2. **Capture token information.** This information will be used to effect payment later. This block is of the same form as (I)-namely. a signed, encrypted capture token together with a digital envelope. This token is not processed by the merchant. Rather, it must be returned, as is, with a payment request.
3. **Certificate.** The gateway's signature key certificate.

With the authorization from the gateway, the merchant can provide the goods or service to the customer.

Payment Capture

To obtain payment, the merchant engages the payment gateway in a payment capture transaction, consisting of a capture request and a capture response message.

For the **Capture Request** message, the merchant gencrates. signs, and encrypts a capture request block, which includes the payment amount and the transaction ID. The message also includes the encrypted capture token received earlier (in the Authorization Response) for this transaction, as well as the merchant's signature key and key-exchange key certificates.

When the payment gateway receives the capture request message. it decrypts and verifies the capture request block and decrypts and verifies the capture token block. It then checks for consistency between the capture request and capture token. It then creates a clearing request that is sent to the issuer over the private payment network. This request causes funds to be transferred to the merchant's account.

The gateway then notifies the merchant of payment in a **Capture Response** message. The message includes a capture response block that the gateway signs and encrypts. The message also includes the gateway's signature key certificate. The merchant software stores the capture response to be used for reconciliation with payment received from the acquirer.

Content beyond Syllabus

- Functional encryption
- Black-Box Impossibility Results
- lattice-based cryptography and foundations

Functional encryption (FE) is a generalization of public-key encryption in which possessing a secret key allows one to learn a function of what the ciphertext is encrypting.

Functional encryption generalizes several existing primitives including Identity-based encryption (IBE) and attribute-based encryption (ABE). In the IBE case, define to be equal to when corresponds to an identity that is allowed to decrypt, and otherwise. Similarly, in the ABE case, define when encodes attributes with permission to decrypt and otherwise.

Black-Box Impossibility Results

As a main **result**, we show that, if a family of collision-resistant hash functions exist, then **black-box** LRZK is **impossible** for non-trivial languages if we only rely on a leak-free input-encoding phase (i.e., without CRS/preprocessing).

lattice-based cryptography, and foundations

These are **cryptographic** schemes whose security relies on the presumed hardness of certain computational problems over ubiquitous (and beautiful) geometric objects called **lattices**.

Lattice-based cryptography is the generic term for constructions of cryptographic primitives that involve lattices, either in the construction itself or in the security proof. Lattice-based constructions are currently important candidates for post-quantum cryptography. Unlike more widely used and known public-key schemes such as the RSA, Diffie-Hellman or elliptic-curve cryptosystems, which could, theoretically, be easily attacked by a quantum computer, some lattice-based constructions appear to be resistant to attack by both classical and quantum computers. Furthermore, many lattice-based constructions are considered to be secure under the assumption that certain well-studied computational lattice problems cannot be solved efficiently.