DS4101PC: Information Security

Mr. V. Anil Kumar
Assistant Professor
Department of IT&DS

References

TextBooks:

1. Cryptography and Network Security - Principles and Practice: William Stallings, Pearson Education, 6th Edition

ReferenceBooks:

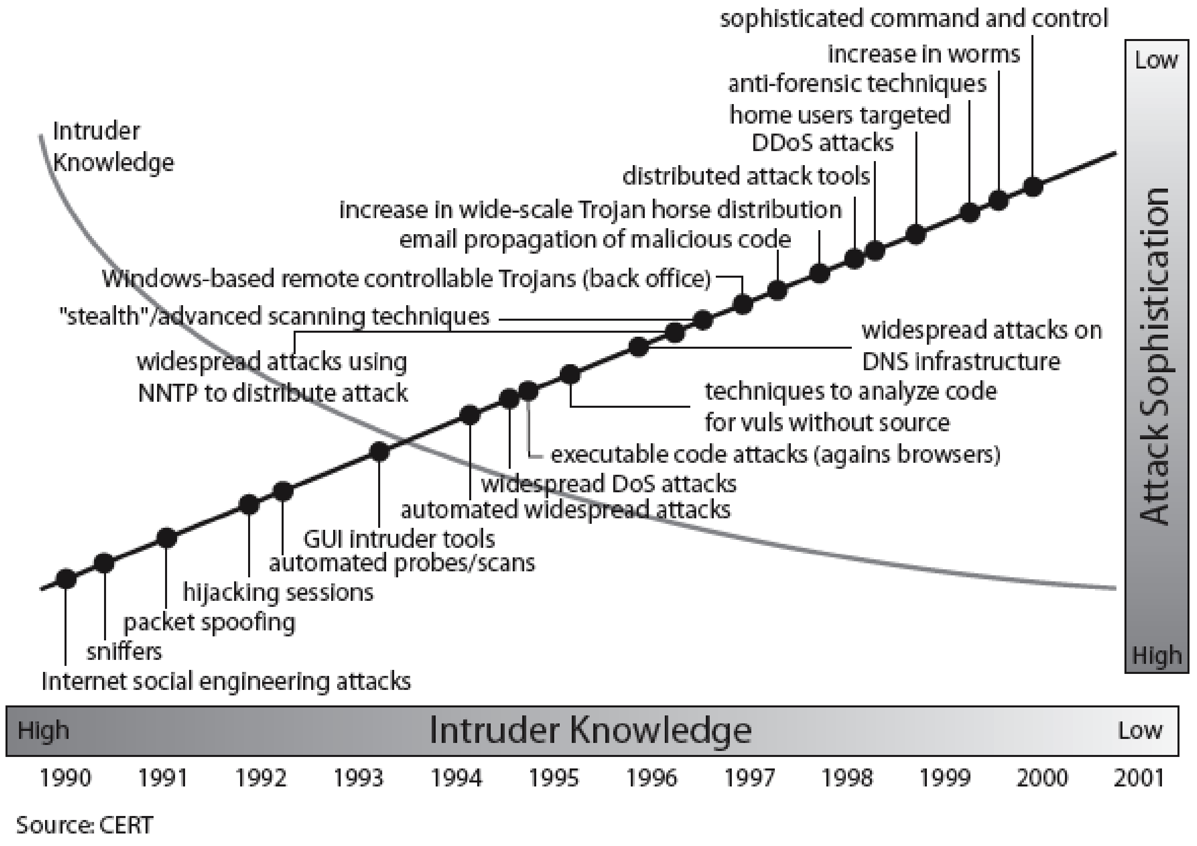
1. Cryptography and Network Security: C K Shyamala, N Harini, Dr T R Padmanabhan, Wiley India, 1st Edition.
2. Cryptography and Network Security: Atul Kahate, Mc Graw Hill, 3rd Edition

Content

Introduction

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

Security Trends



Services, Mechanisms, Attacks

- need systematic way to define requirements
- consider three aspects of information security:
  - **security attack**
  - **security mechanism**
  - **security service**
- consider in reverse order

Security Service

- is something that enhances the security of the data processing systems and the information transfers of an organization
- intended to counter security attacks
- make use of one or more security mechanisms to provide the service
- replicate functions normally associated with physical documents

  eg have signatures, dates; need protection from disclosure, tampering, or destruction; be notarized or witnessed; be recorded or licensed

Security Mechanism

- a mechanism that is designed to detect, prevent, or recover from a security attack
- no single mechanism that will support all functions required
- however one particular element underlies many of the security mechanisms in use: **cryptographic techniques**
- hence our focus on this area

Security Attack

- any action that compromises the security of information owned by an organization
- information security is about how to prevent attacks, or failing that, to detect attacks on information-based systems
- have a wide range of attacks
- can focus of generic types of attacks
- note: often *threat* & *attack* mean same

Security Services

- X.800 defines it as: a service provided by a protocol layer of communicating open systems, which ensures adequate security of the systems or of data transfers
- RFC 2828 defines it as: a processing or communication service provided by a system to give a specific kind of protection to system resources
- X.800 defines it in 5 major categories

Security Services (X.800)

- **Authentication** - assurance that the communicating entity is the one claimed
- **Access Control** - prevention of the unauthorized use of a resource
- **Data Confidentiality** –protection of data from unauthorized disclosure
- **Data Integrity** - assurance that data received is as sent by an authorized entity
- **Non-Repudiation** - protection against denial by one of the parties in a communication

Security Mechanisms (X.800)

specific security mechanisms:

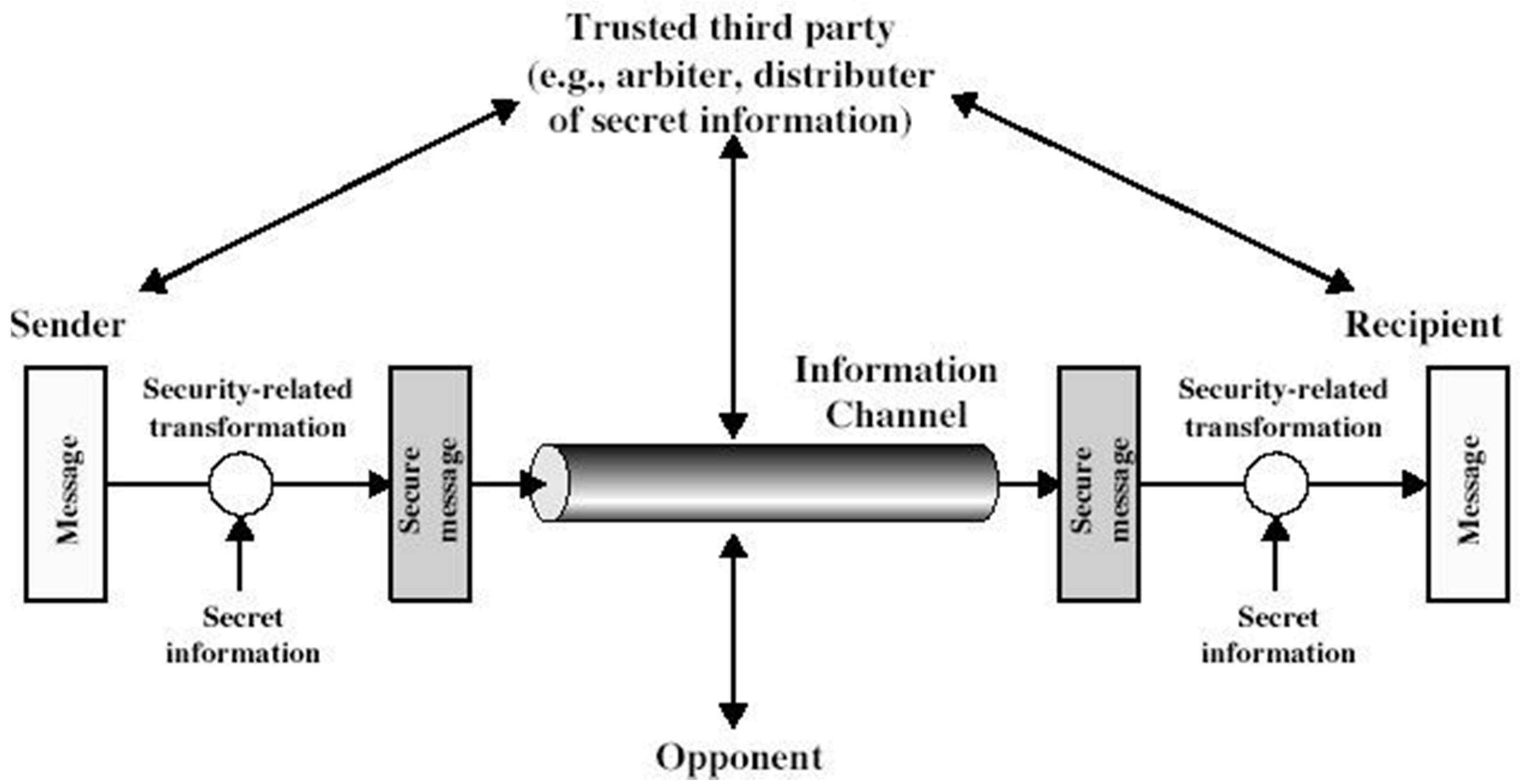
- encipherment, digital signatures, access controls, data integrity, authentication exchange, traffic padding, routing control, notarization

- pervasive security mechanisms:
  - trusted functionality, security labels, event detection, security audit trails, security recovery

Classify Security Attacks as

- **passive attacks** - eavesdropping on, or monitoring of, transmissions to:
  - obtain message contents, or
  - monitor traffic flows
- **active attacks** – modification of data stream to:
  - masquerade of one entity as some other
  - replay previous messages
  - modify messages in transit
  - denial of service

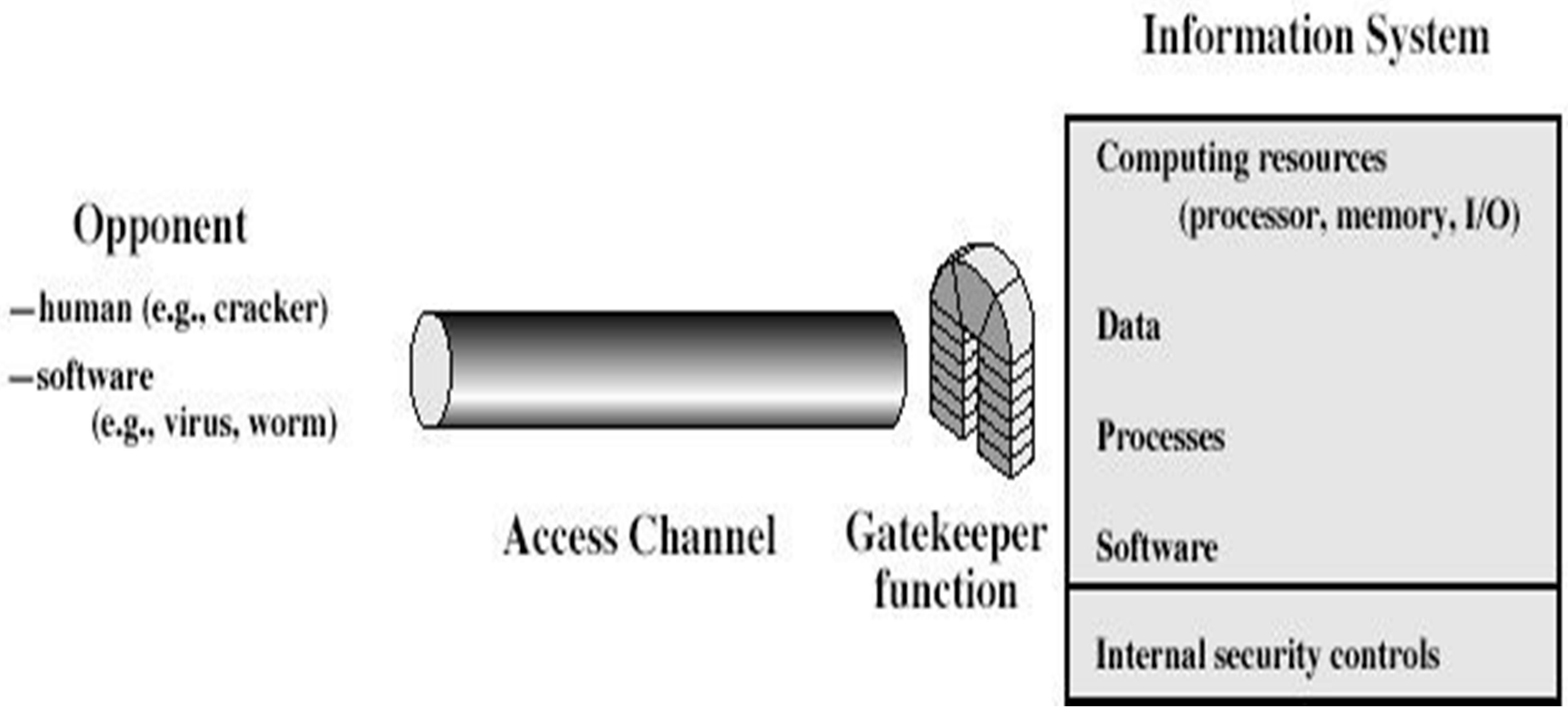
Model for Network Security



Model for Network Security

- using this model requires us to:
  - design a suitable algorithm for the security transformation
  - generate the secret information (keys) used by the algorithm
  - develop methods to distribute and share the secret information
  - specify a protocol enabling the principals to use the transformation and secret information for a security service

Model for Network Access Security



Model for Network Access Security

- using this model requires us to:
  - select appropriate gatekeeper functions to identify users
  - implement security controls to ensure only authorised users access designated information or resources
- trusted computer systems can be used to implement this model

Cryptography and Network Security

Basic Terminology

- **plaintext** - the original message
- **ciphertext** - the coded message
- **cipher** - algorithm for transforming plaintext/ciphertext
- **key** - info used in cipher known only to sender/receiver
- **encipher (encrypt)** - converting plaintext to ciphertext
- **decipher (decrypt)** - recovering ciphertext from plaintext
- **cryptography** - study of encryption principles/methods
- **cryptanalysis (codebreaking)** - the study of principles/ methods of deciphering ciphertext *without* knowing key
- **cryptology** - the field of both cryptography and cryptanalysis

Two kinds of Ciphers

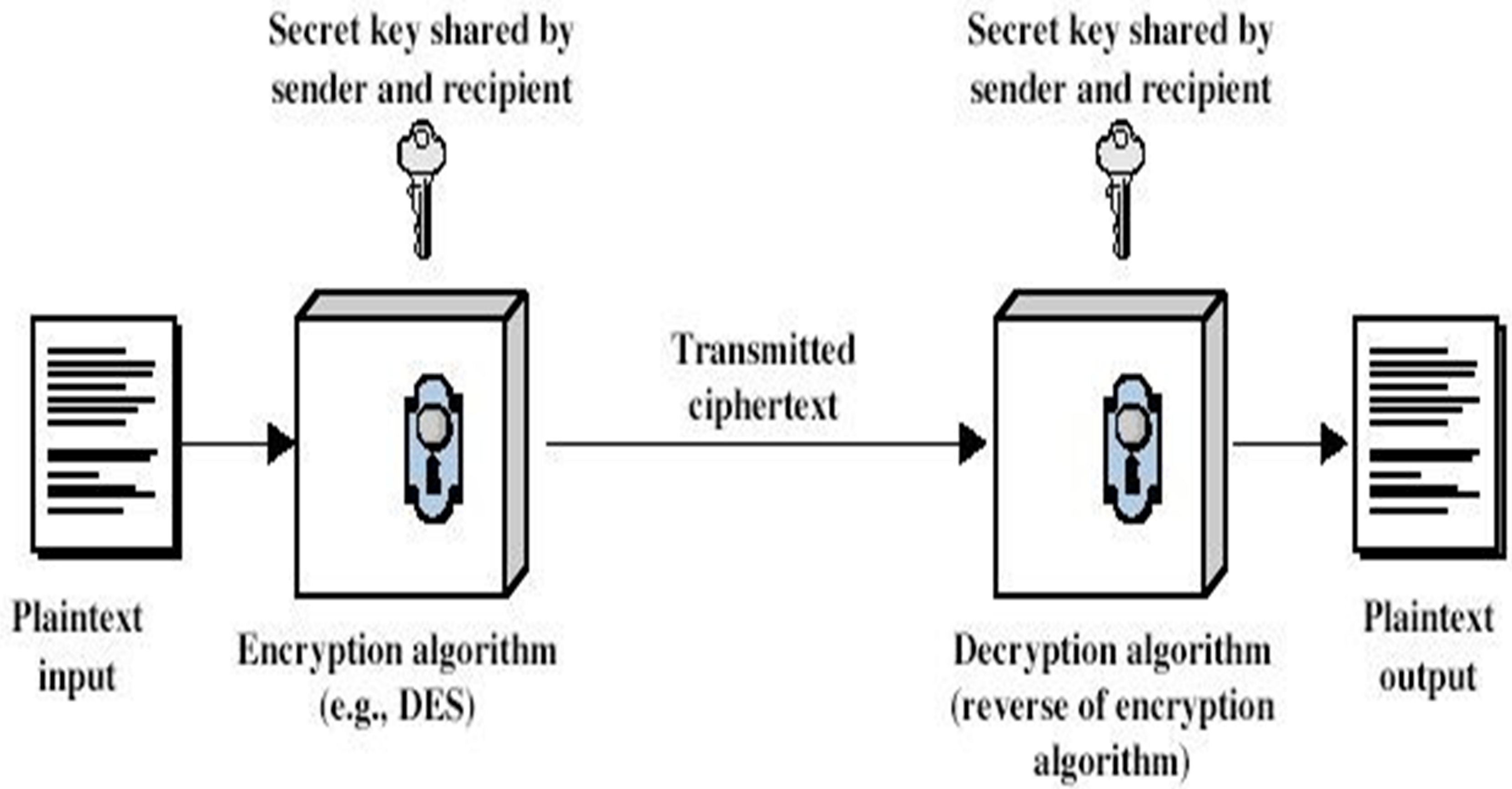
State-of-the-art: two kinds of most popular encryption algorithms

- Symmetric ciphers
  - Sender and receiver share a common key
- Public key ciphers
  - Sender and receiver have asymmetric information of the key(s)

Symmetric Encryption

- or conventional / private-key / single-key
- was only type prior to invention of public-key in 1970's
- remains very widely used
- sender and recipient share a common key
  - Both parties have full information of the key
- all classical encryption algorithms are common key (private-key)
  - Characteristic of conventional algorithms

Symmetric Cipher Model



Secret key shared by sender and recipient
Secret key shared by sender and recipient
Transmitted ciphertext
Plaintext input
Encryption algorithm (e.g., DES)
Decryption algorithm (reverse of encryption algorithm)
Plaintext output

Requirements

- two requirements for secure use of symmetric encryption:
  - a strong encryption algorithm (keeping key secret is sufficient for security)
  - a secret key known only to sender / receiver

    $Y = \mathrm{E}_K(X)$

    $X = \mathrm{D}_K(Y)$
- assume encryption algorithm is known
- implies a secure channel to distribute key

Cryptography

- can characterize by:
  - type of encryption operations used
    - substitution / transposition / product systems
  - number of keys used
    - single-key or private / two-key or public
  - way in which plaintext is processed
    - Block: process one block of elements a time
    - Stream: continuous input, output one element a time

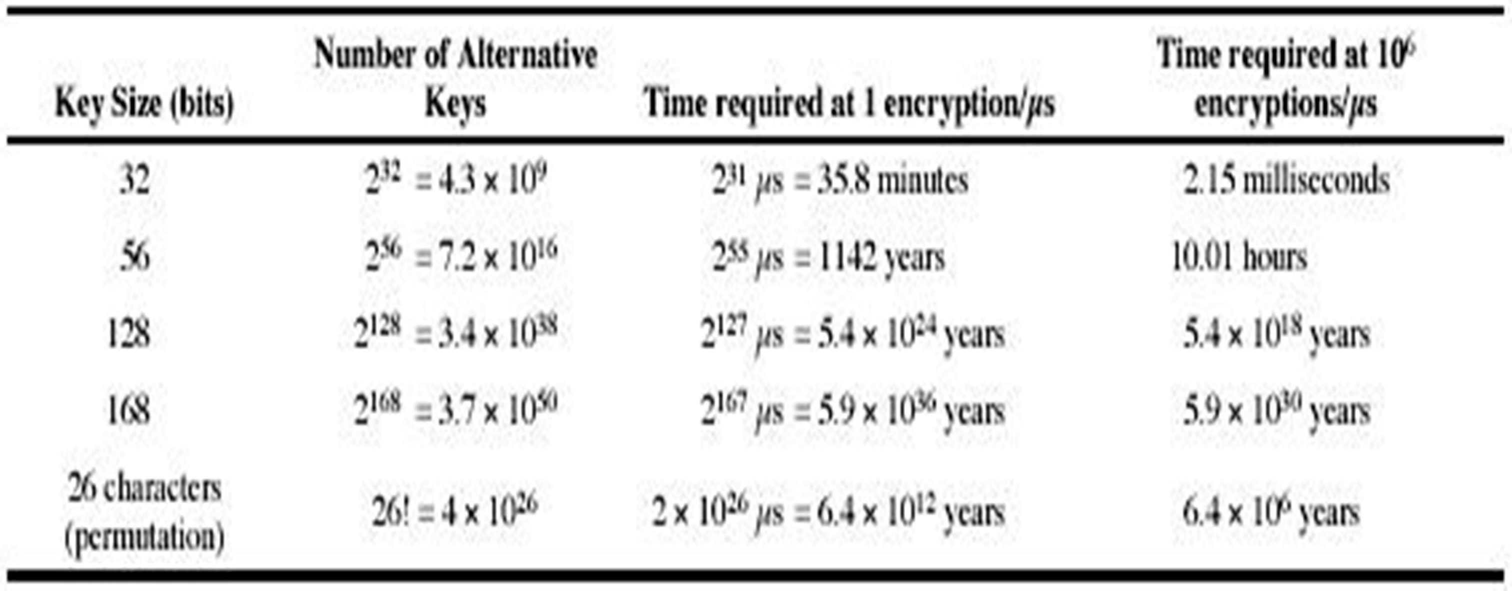
Types of Cryptanalytic Attacks

- **ciphertext only**
  - know a) algorithm b) ciphertext
- **known plaintext**
  - know some given plaintext/ciphertext pairs
- **chosen plaintext**
  - select plaintext and obtain ciphertext
- **chosen ciphertext**
  - select ciphertext and obtain plaintext

- **chosen text**
  - select either plaintext or ciphertext to en/decrypt to attack cipher

Brute Force Search

- always possible to simply try every key
- most basic attack, proportional to key size
- assume either know / recognise plaintext



Key Size (bits)	Number of Alternative Keys	Time required at 1 encryption/μs	Time required at 10^6 encryptions/μs
32	$2^{32} = 4.3 \times 10^9$	$2^{31}\ \mu s = 35.8$ minutes	2.15 milliseconds
56	$2^{56} = 7.2 \times 10^{16}$	$2^{55}\ \mu s = 1142$ years	10.01 hours
128	$2^{128} = 3.4 \times 10^{38}$	$2^{127}\ \mu s = 5.4 \times 10^{24}$ years	5.4×10^{18} years
168	$2^{168} = 3.7 \times 10^{50}$	$2^{167}\ \mu s = 5.9 \times 10^{36}$ years	5.9×10^{30} years
26 characters (permutation)	$26! = 4 \times 10^{26}$	$2 \times 10^{26}\ \mu s = 6.4 \times 10^{12}$ years	6.4×10^6 years

More Definitions

- **unconditional security**
  - no matter how much computer power is available, the cipher cannot be broken since the ciphertext provides insufficient information to uniquely determine the corresponding plaintext (non-exist in real applications)
- **computational security**
  - given limited computing resources (eg time needed for calculations is greater than age of universe), the cipher cannot be broken

Classical Ciphers

- Examine a sampling of what might be called classical encryption techniques.
- Illustrate the basic approaches to symmetric encryption and the types of cryptanalytic attacks that must be anticipated.
- The two basic building blocks of all encryption techniques: substitution and transposition.

Classical Substitution Ciphers

- where letters of plaintext are replaced by other letters or by numbers or symbols
- or if plaintext is viewed as a sequence of bits, then substitution involves replacing plaintext bit patterns with ciphertext bit patterns

Caesar Cipher

- earliest known substitution cipher
- by Julius Caesar
- first attested use in military affairs
- replaces each letter by a letter *three* places down the alphabet
- example:

  meet me after the toga party

  PHHW PH DIWHU WKH WRJD SDUWB

Caesar Cipher

- can define transformation as:

  a b c d e f g h i j k l m n o p q r s t u v w x y z

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

- mathematically give each letter a number

a	b	c	d	e	f	g	h	i	j	k	l	m
0	1	2	3	4	5	6	7	8	9	10	11	12

n	o	p	q	r	s	t	u	v	w	x	y	Z
13	14	15	16	17	18	19	20	21	22	23	24	25

- then have Caesar cipher as:

  $C = E(p) = (p + k) \bmod (26)$

  $p = D(C) = (C - k) \bmod (26)$

  – **modulo arithmetic:** 1 = 27 mod 26, 3 = 29 mod 26

Cryptanalysis of Caesar Cipher

- only have 26 possible keys
  - Could shift K = 0, 1, 2, ..., 25 slots
- could simply try each in turn
- a **brute force search**
- given ciphertext, just try all shifts of letters
- do need to recognize when have plaintext
- Test:break ciphertext

  GCUA VQ DTGCM

Monoalphabetic Cipher

- rather than just shifting the alphabet
- could shuffle the letters arbitrarily
- each plaintext letter maps to a different random ciphertext letter
- hence key is 26 letters long

Plain: abcdefghijklmnopqrstuvwxyz
Cipher: DKVQFIBJWPESCXHTMYAUoLRGZN
Plaintext: ifwewishtoreplaceletters
Ciphertext: WIRFRWAJUHYFTSDVFSFUUFYA

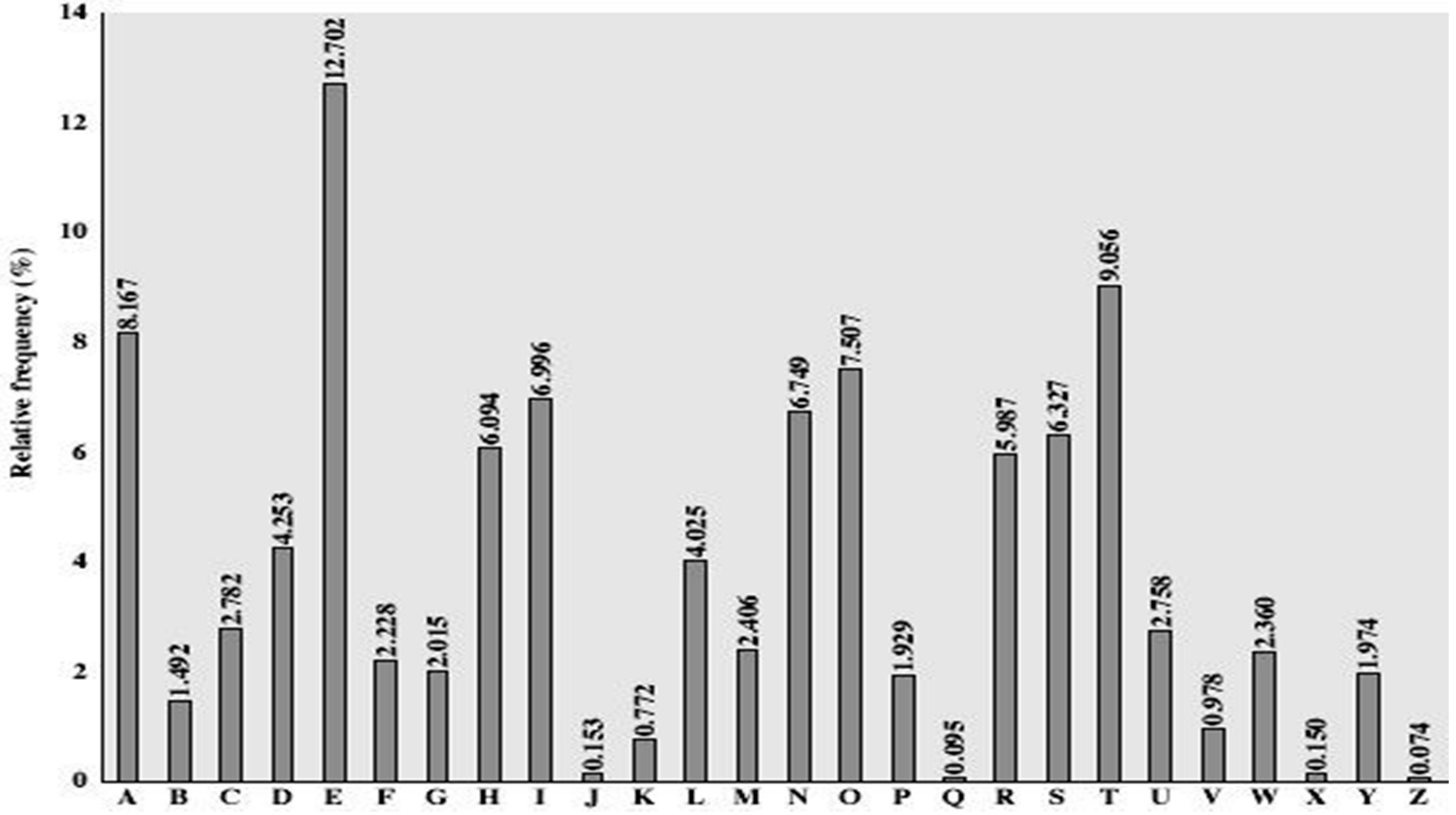
Monoalphabetic Cipher Security

- now have a total of 26! = 4 x 10^26 keys
- with so many keys, might think is secure
  - The simplicity and strength of the monoalphabetic substitution cipher dominated for the first millenium AD.
- but would be **!!!WRONG!!!**
  - First broken by Arabic scientists in 9th century

Frequency Analysis

- letters are not equally commonly used
- in English **e** is by far the most common letter
- then T,R,N,I,O,A,S
- other letters are fairly rare
- cf. Z,J,K,Q,X
- have tables of single, double & triple letter frequencies

English Letter Frequencies



Use in Cryptanalysis

- key concept - monoalphabetic substitution ciphers do not change relative letter frequencies
- discovered by Arabian scientists in 9th century
- calculate letter frequencies for ciphertext
- compare counts/plots against known values
- for monoalphabetic must identify each letter
  - tables of common double/triple letters help

Example Cryptanalysis

- given ciphertext:

  UZQSoVUoHXMoPVGPoZPEVSGZWSZoPFPESXUDBMETSXAIZ
  VUEPHZHMDZSHZoWSFPAPPDTSVPQUZWYMXUZUHSX
  EPYEPoPDZSZUFPoMBZWPFUPZHMDJUDTMoHMQ
- count relative letter frequencies (see text)
- guess P & Z are e and t
- guess ZW is th and hence ZWP is the
- proceeding with trial and error finally get:

  it was disclosed yesterday that several informal but
  direct contacts have been made with political
  representatives of the viet cong in moscow

Playfair Cipher

- not even the large number of keys in a monoalphabetic cipher provides security
- one approach to improving security was to encrypt multiple letters
- the **Playfair Cipher** is an example
- invented by Charles Wheatstone in 1854, but named after his friend Baron Playfair

Playfair Key Matrix

- a 5X5 matrix of letters based on a keyword
- fill in letters of keyword (sans duplicates)
- fill rest of matrix with other letters
- eg. using the keyword MONARCHY

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

Encrypting and Decrypting

- plaintext encrypted two letters at a time:
  1. if a pair is a repeated letter, insert a filler like 'X', eg. "balloon" encrypts as "ba lx lo on"
  2. if both letters fall in the same row, replace each with letter to right (wrapping back to start from end), eg. “ar" encrypts as "RM"
  3. 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"
  4. 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 of the Playfair Cipher

- security much improved over monoalphabetic
- since have 26 x 26 = 676 digrams
- would need a 676-entry frequency table to analyse (verses 26 for a monoalphabetic)
- and correspondingly more ciphertext
- was widely used for many years (eg. US & British military in WW1)
- it **can** be broken, given a few hundred letters

Polyalphabetic Ciphers

- another approach to improving security is to use multiple cipher alphabets
- called **polyalphabetic substitution ciphers**
- makes cryptanalysis harder with more alphabets to guess and flatter frequency distribution
- use a key to select which alphabet is used for each letter of the message
- use each alphabet in turn
- repeat from start after end of key is reached

Example

key: deceptivedeceptivedeceptive

plaintext: wearediscoveredsaveyourself

ciphertext:ZICVTWQNGRZGVTWAVZHCQYGLMGJ

- write the plaintext out
- write the keyword repeated above it
  - eg using keyword *deceptive*
- use each key letter as a caesar cipher key
- encrypt the corresponding plaintext letter

Vigenère Cipher

- simplest polyalphabetic substitution cipher is the **Vigenère Cipher**
- effectively multiple caesar ciphers
- key is d-letter long K = k1 k2 ... kd
- i^{th} letter specifies i^{th} alphabet to use
- use each alphabet in turn
- repeat from start after d letters in message
- decryption simply works in reverse

Security of Vigenère Ciphers

- have multiple ciphertext letters for each plaintext letter
- hence letter frequencies are obscured
- but not totally lost
- start with letter frequencies
  - see if look monoalphabetic or not

Transposition Ciphers

- now consider classical **transposition** or **permutation** ciphers
- these hide the message by rearranging the letter order
- without altering the actual letters used
- can recognise these since have the same frequency distribution as the original text

Rail Fence cipher

- write message letters out diagonally over a number of rows
- then read off cipher row by row
- eg. write message out as:

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

- giving ciphertext

  MEMATRHTGPRYETEFETEoAAT

Row Transposition Ciphers

- a more complex scheme
- write letters of message out in rows over a specified number of columns
- then reorder the columns according to some key before reading off the rows

```
Key:        4 3 1 2 5 6 7
Plaintext:  a t t a c k p
            o s t p o n e
            d u n t i l t
            w o a m x y z
Ciphertext:   TTNAAPTMTSUoAoDWCoIXKNLYPETZ
```

Product Ciphers

- ciphers using substitutions or transpositions are not secure because of language characteristics
- hence consider using several ciphers in succession to make harder, but:
  - two substitutions make a more complex substitution
  - two transpositions make more complex transposition
  - but a substitution followed by a transposition makes a new much harder cipher
- this is bridge from classical to modern ciphers

Rotor Machines

- Multiple-stage substitution algorithms
- before modern ciphers, rotor machines were most common product cipher
- were widely used in WW2
  - German Enigma, Allied Hagelin, Japanese Purple
- implemented a very complex, varying substitution cipher
- used a series of cylinders, each giving one substitution, which rotated and changed after each letter was encrypted

Steganography

- an alternative to encryption
- hides existence of message
  – using only a subset of letters/words in a longer message marked in some way
  – using invisible ink
  – hiding graphic image or sound file
- has drawbacks
  – high overhead to hide relatively few info bits

Block vs Stream Ciphers

- block ciphers process messages into blocks, each of which is then en/decrypted
- like a substitution on very big characters
  - 64-bits or more
- stream ciphers process messages a bit or byte at a time when en/decrypting
- many current ciphers are block ciphers
- hence are focus of course

Block Cipher Principles

- block ciphers look like an extremely large substitution
- would need table of 2^{64} entries for a 64-bit block
- arbitrary reversible substitution cipher for a large block size is not practical
  - 64-bit general substitution block cipher, key size 2^{64}!
- most symmetric block ciphers are based on a **Feistel Cipher Structure**
- needed since must be able to **decrypt** ciphertext to recover messages efficiently

C. Shannon and Substitution-Permutation Ciphers

- in 1949 Shannon introduced idea of substitution-permutation (S-P) networks
  - modern substitution-transposition product cipher
- these form the basis of modern block ciphers
- S-P networks are based on the two primitive cryptographic operations we have seen before:
  - *substitution* (S-box)
  - *permutation* (P-box) (transposition)
- provide *confusion* and *diffusion* of message

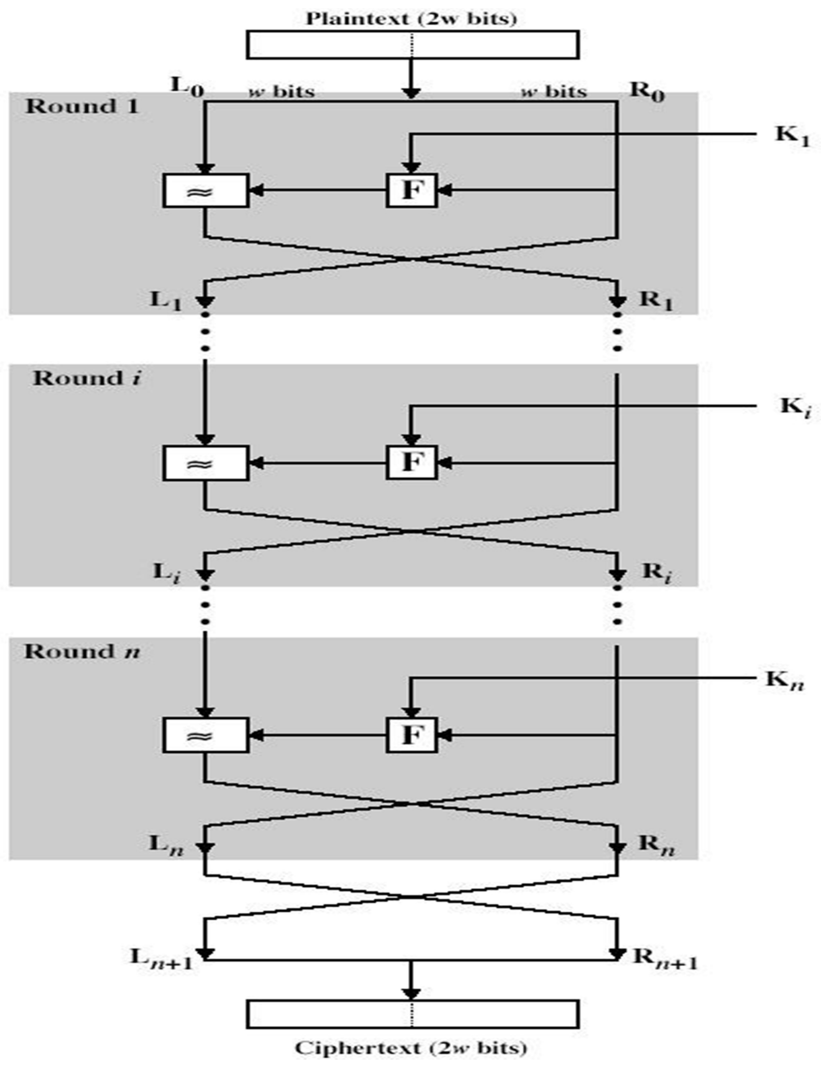
Diffusion and Confusion

- Introduced by Claude Shannon to thwart cryptanalysis based on statistical analysis – Assume the attacker has some knowledge of
  - the statistical characteristics of the plaintext
- cipher needs to completely obscure statistical properties of original message
- a one-time pad does this

Feistel Cipher Structure

- Horst Feistel devised the **feistel cipher**
  - implements Shannon's substitution-permutation network concept
- partitions input block into two halves
  - process through multiple rounds which
  - perform a substitution on left data half
  - based on round function of right half & subkey
  - then have permutation swapping halves

Feistel Cipher Structure



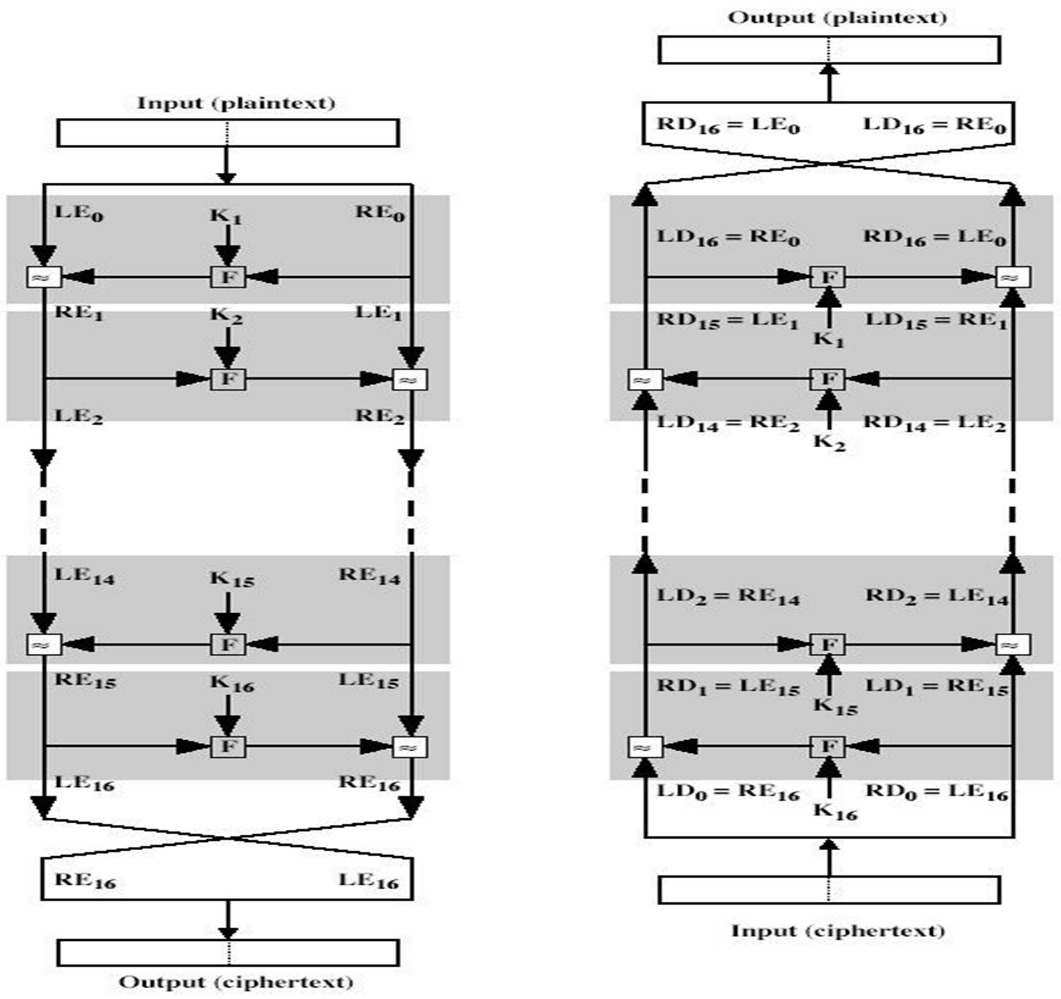
Feistel Cipher

- n sequential rounds
- A substitution on the left half L_i
  - 1. Apply a round function F to the right half R_i and
  - 2. Take XOR of the output of (1) and L_i
- The round function is parameterized by the subkey K_i
  - K_i are derived from the overall key K

Feistel Cipher Design Principles

- **block size**
  - increasing size improves security, but slows cipher
- **key size**
  - increasing size improves security, makes exhaustive key searching harder, but may slow cipher
- **number of rounds**
  - increasing number improves security, but slows cipher
- **subkey generation**
  - greater complexity can make analysis harder, but slows cipher
- **round function**
  - greater complexity can make analysis harder, but slows cipher
- **fast software en/decryption & ease of analysis**
  - are more recent concerns for practical use and testing

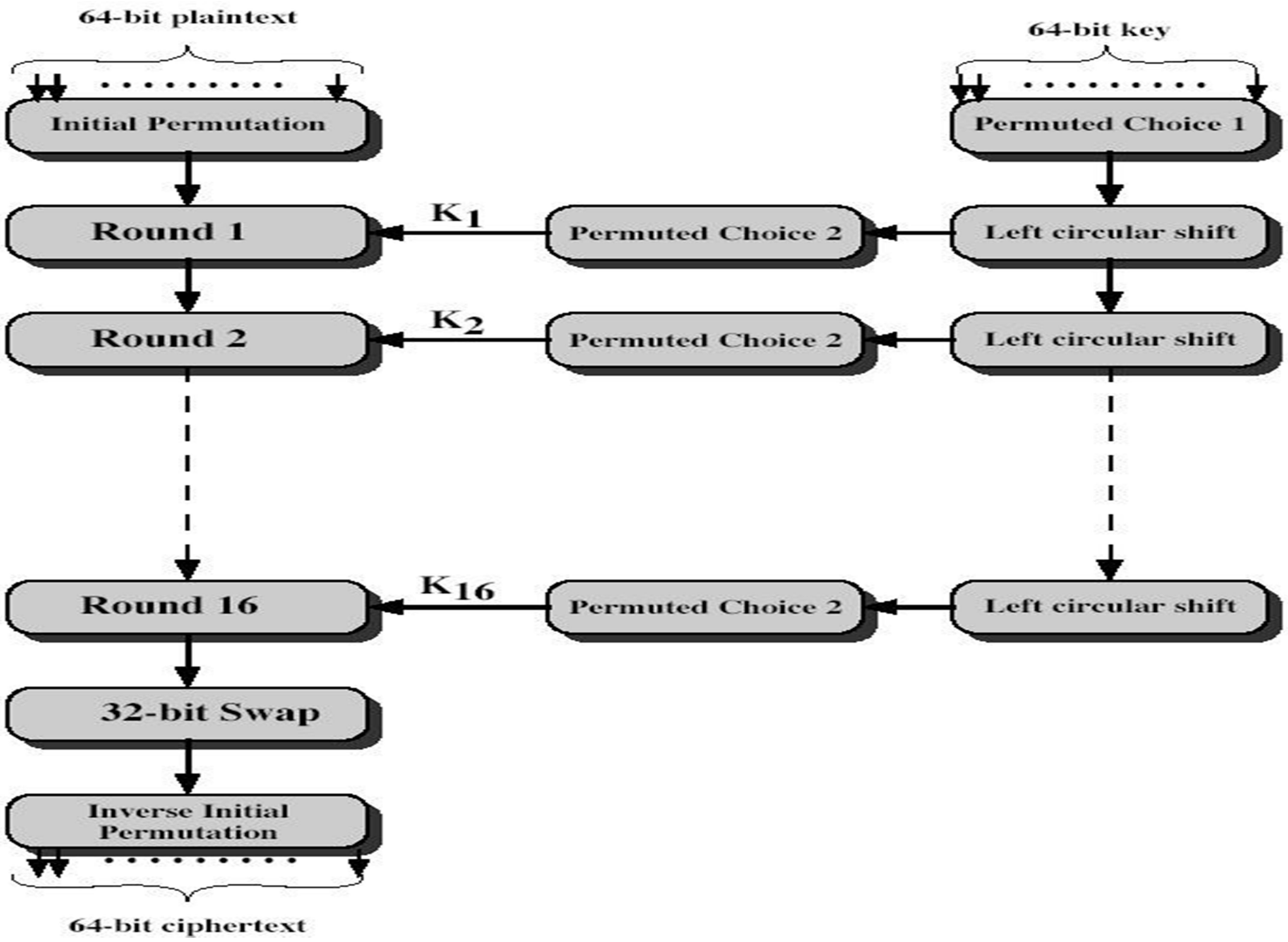
Feistel Cipher Decryption



Data Encryption Standard (DES)

- most widely used block cipher in world
- adopted in 1977 by NBS (now NIST)
  - as FIPS PUB 46
- encrypts 64-bit data using 56-bit key
- has widespread use

DES Encryption



Initial Permutation IP

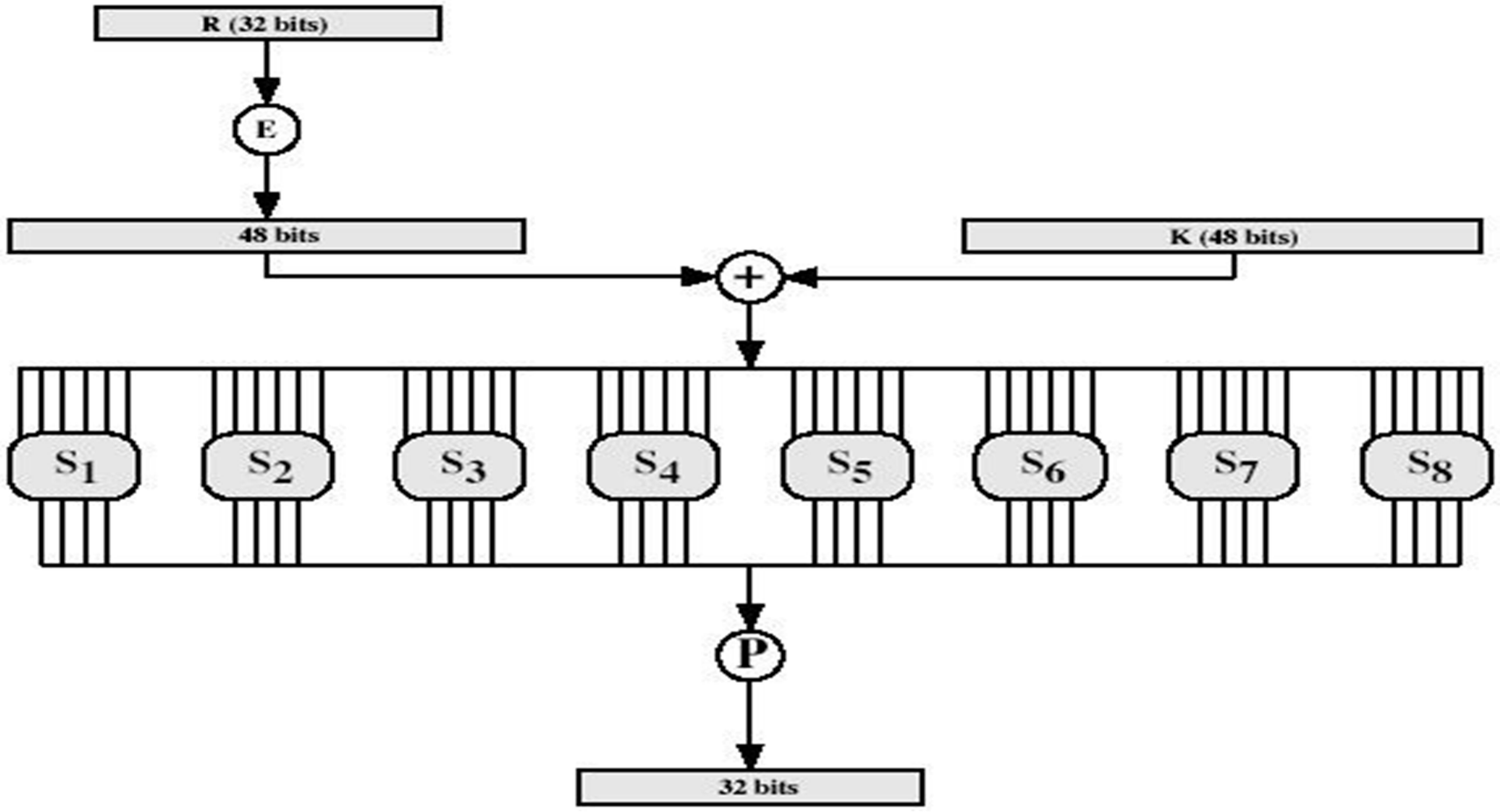
- first step of the data computation
- IP reorders the input data bits
- quite regular in structure
  - see text Table 3.2
- example:

ıP(675a6967 5e5a6b5a) = (ffb2194d 004df6fb)

DES Round Structure

- uses two 32-bit L & R halves
- as for any Feistel cipher can describe as:
  $L_i = R_{i-1}$
  $R_i = L_{i-1}$ xor $F(R_{i-1}, K_i)$
- takes 32-bit R half and 48-bit subkey and:
  - expands R to 48-bits using Expansion Permutation E (Table 3.2 c.)
  - adds to subkey
  - passes through 8 S-boxes to get 32-bit result
  - finally permutes this using 32-bit Permutation Function P (Table 3.2 d)

The round function F(R,K)



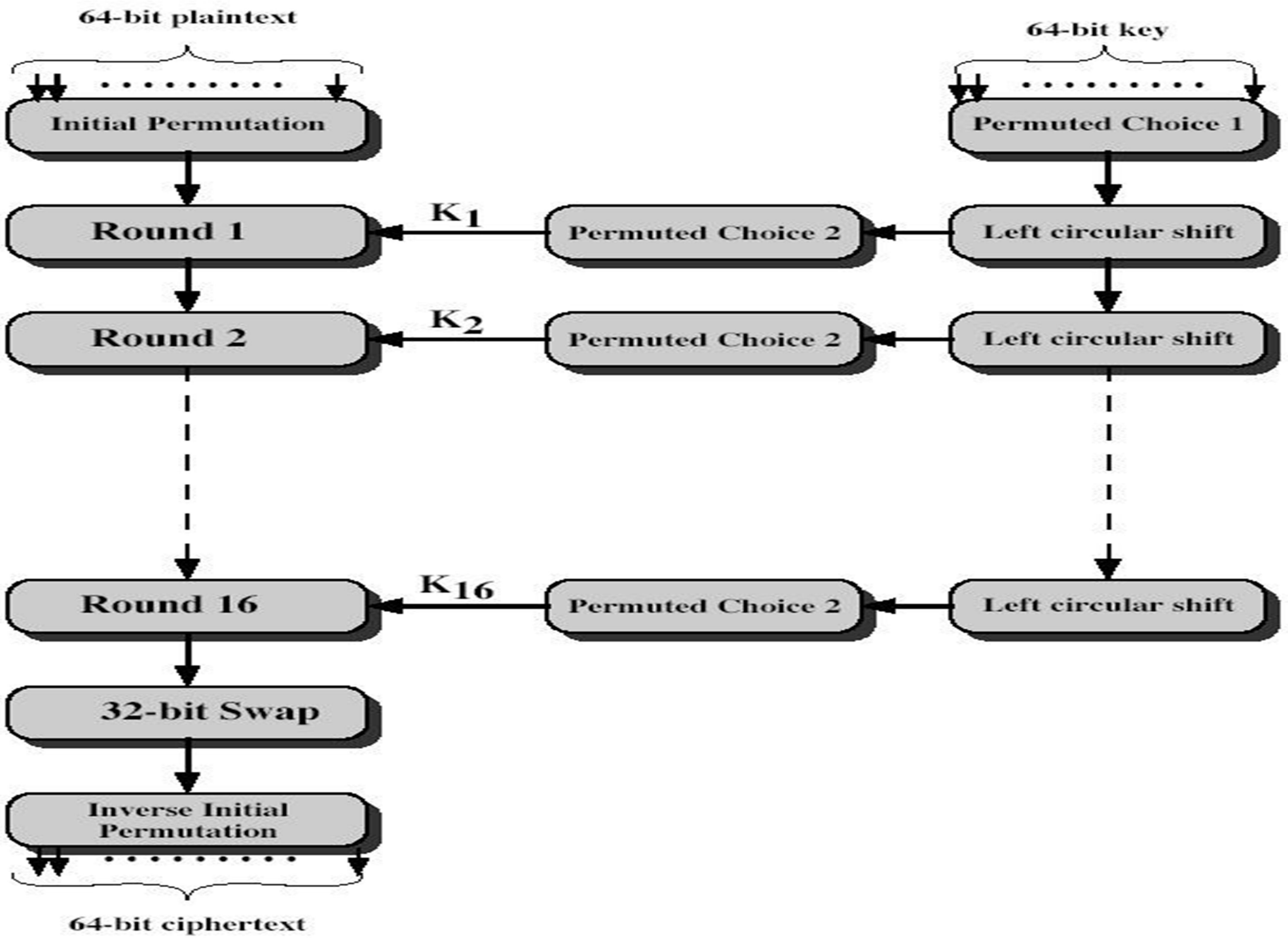
Substitution Boxes S

- 8 S-boxes (Table 3.3)
- Each S-Box mapps 6 to 4 bits
  - outer bits 1 & 6 (**row** bits) select the row
  - inner bits 2-5 (**col** bits) select the column
  - For example, in S1, for input 011001,
    - the row is 01 (row 1)
    - the column is 1100 (column 12).
    - The value in row 1, column 12 is 9
    - The output is 1001.
- result is 8 X 4 bits, or 32 bits

DES Decryption

- decrypt must unwind steps of data computation
- with Feistel design, do encryption steps again
- using subkeys in reverse order (SK16 … SK1)
- note that IP undoes final FP step of encryption
- 1st round with SK16 undoes 16th encrypt round
- ....
- 16th round with SK1 undoes 1st encrypt round
- then final FP undoes initial encryption IP
- thus recovering original data value

DES Decryption (reverse encryption)



Avalanche Effect

- key desirable property of encryption alg
- DES exhibits strong avalanche
- where a change of **one** input or key bit results in changing approx **half** output bits

Strength of DES - Key Size

- 56-bit keys have $2^{56} = 7.2 \times 10^{16}$ values
- brute force search looks hard
- recent advances have shown is possible
  - in 1997 on Internet in a few months
  - in 1998 on dedicated hardware (EFF) in a few days
  - in 1999 above combined in 22hrs!
- still must be able to recognize plaintext

Strength of DES - Timing Attacks

- attacks actual implementation of cipher
- use knowledge of consequences of implementation to derive knowledge of some/all subkey bits
- specifically use fact that calculations can take varying times depending on the value of the inputs to it

Strength of DES - Analytic Attacks

- now have several analytic attacks on DES
- these utilise some deep structure of the cipher
  - by gathering information about encryptions
  - can eventually recover some/all of the sub-key bits
  - if necessary then exhaustively search for the rest
- generally these are statistical attacks
- include
  - differential cryptanalysis
  - linear cryptanalysis

Differential Cryptanalysis

- one of the most significant recent (public) advances in cryptanalysis
- known in 70's with DES design
- Murphy, Biham & Shamir published 1990
- powerful method to analyse block ciphers
- used to analyse most current block ciphers with varying degrees of success
- DES reasonably resistant to it

Differential Cryptanalysis

- a statistical attack against Feistel ciphers
- uses cipher structure not previously used
- design of S-P networks has output of function *f* influenced by both input & key
- hence cannot trace values back through cipher without knowing values of the key

Differential Cryptanalysis Compares Pairs of Encryptions

- Differential cryptanalysis is complex
- with a known difference in the input
- searching for a known difference in output

$$\Delta m_{i+1} = m_{i+1} \oplus m'_{i+1}$$

$$= \left[m_{i-1} \oplus \mathrm{f}(m_i, K_i)\right] \oplus \left[m'_{i-1} \oplus \mathrm{f}(m'_i, K_i)\right]$$

$$= \Delta m_{i-1} \oplus \left[\mathrm{f}(m_i, K_i) \oplus \mathrm{f}(m'_i, K_i)\right]$$

Differential Cryptanalysis

- have some input difference giving some output difference with probability p
- if find instances of some higher probability input / output difference pairs occurring
- can infer subkey that was used in round
- then must iterate process over many rounds

Differential Cryptanalysis

- perform attack by repeatedly encrypting plaintext pairs with known input XOR until obtain desired output XOR
- when found
  - if intermediate rounds match required XOR have a **right pair**
  - if not then have a **wrong pair**
- can then deduce keys values for the rounds
  - right pairs suggest same key bits
  - wrong pairs give random values
- larger numbers of rounds makes it more difficult
- Attack on full DES requires an effort on the order of 2^{47}, requiring 2^{47} chosen plaintexts to be encrypted

Linear Cryptanalysis

- another recent development
- also a statistical method
- based on finding linear approximations to model the transformation of DES
- can attack DES with 2^{47} known plaintexts, still in practise infeasible

Criteria for S-Boxes

- No output of any S-Box is too close to a linear function of the input bits
- Each row of an S-Box includes all 16 possible output bit combinations
- If two inputs to an S-box differ in one bit, the output bits differ in at least two bits
- If two inputs differ is the two middle bits, outputs must differ at least two bits
- Defend against differential analysis and provide good confusion properties

Block Cipher Design Principles

- basic principles still like Feistel in 1970’s
- number of rounds
  - more is better, makes exhaustive search best attack
  - 16 rounds: brute force 2^{55}
  - differential analysis: $2^{55.1}$

Block Cipher Design Principles

- function F:
  - provides “confusion”, is nonlinear, avalanche
  - Strict Avalanche Criterion (SAC)
    - Any output bit i should change with p=1/2 when any single input bit j is inverted, for all i,j
    - Applies to both S-Boxes and the overall F function
- key schedule
  - No general rule has been discovered

Modes of Operation

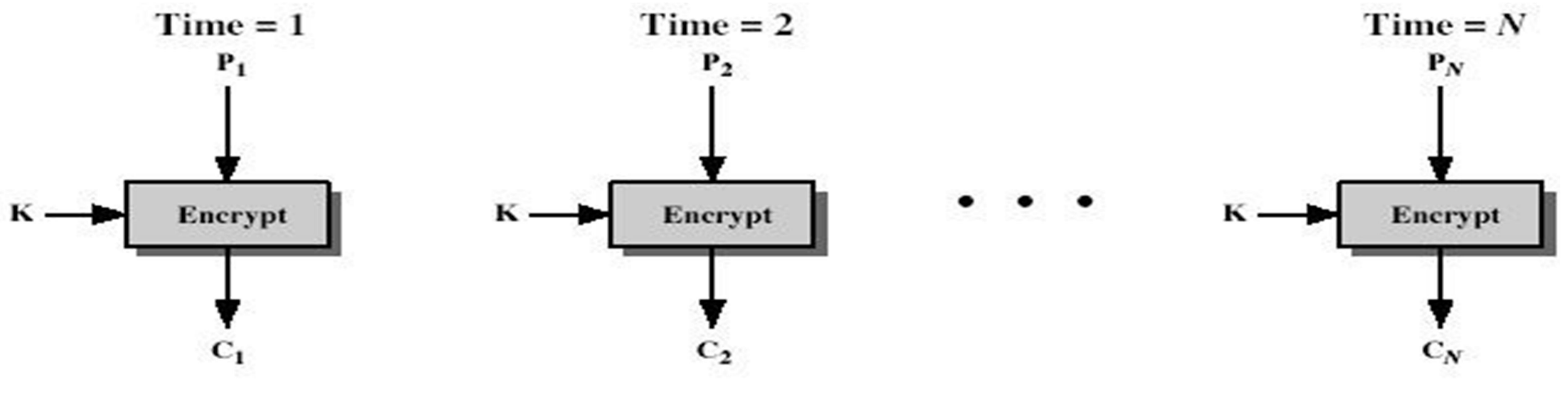
- block ciphers encrypt fixed size blocks
- eg. DES encrypts 64-bit blocks, with 56-bit key
- need way to use in practise, given usually have arbitrary amount of information to encrypt
- four were defined for DES in ANSI standard **ANSI X3.106-1983 Modes of Use**
  - DES is the basic building block
- have **block** and **stream** modes

Electronic Codebook Book (ECB)

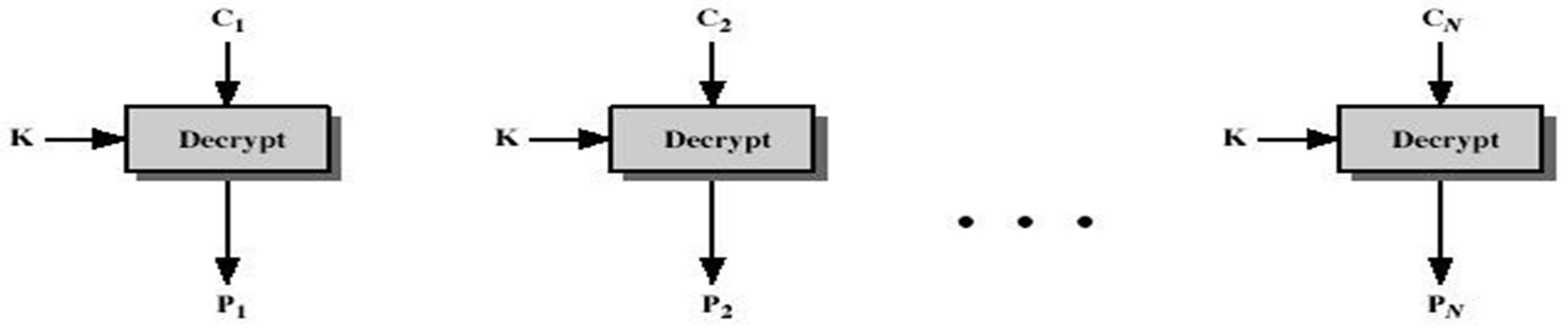
- message is broken into independent blocks which are encrypted
- each block is a value which is substituted, like a codebook, hence name
  - Each DES is a very complex 64-bit to 64-bit substitution
- each block is encoded independently of the other blocks

  $C_i = DES_{K1}(P_i)$
- uses: secure transmission of single values
  - Repeated input blocks have same output
  - Not secure for long transmission

Electronic Codebook Book (ECB)



(a) Encryption



(b) Decryption

Advantages and Limitations of ECB

- repetitions in message may show in ciphertext
  - if aligned with message block
  - particularly with data such graphics
  - or with messages that change very little, which become a code-book analysis problem
- weakness due to encrypted message blocks being independent
- main use is sending a few blocks of data

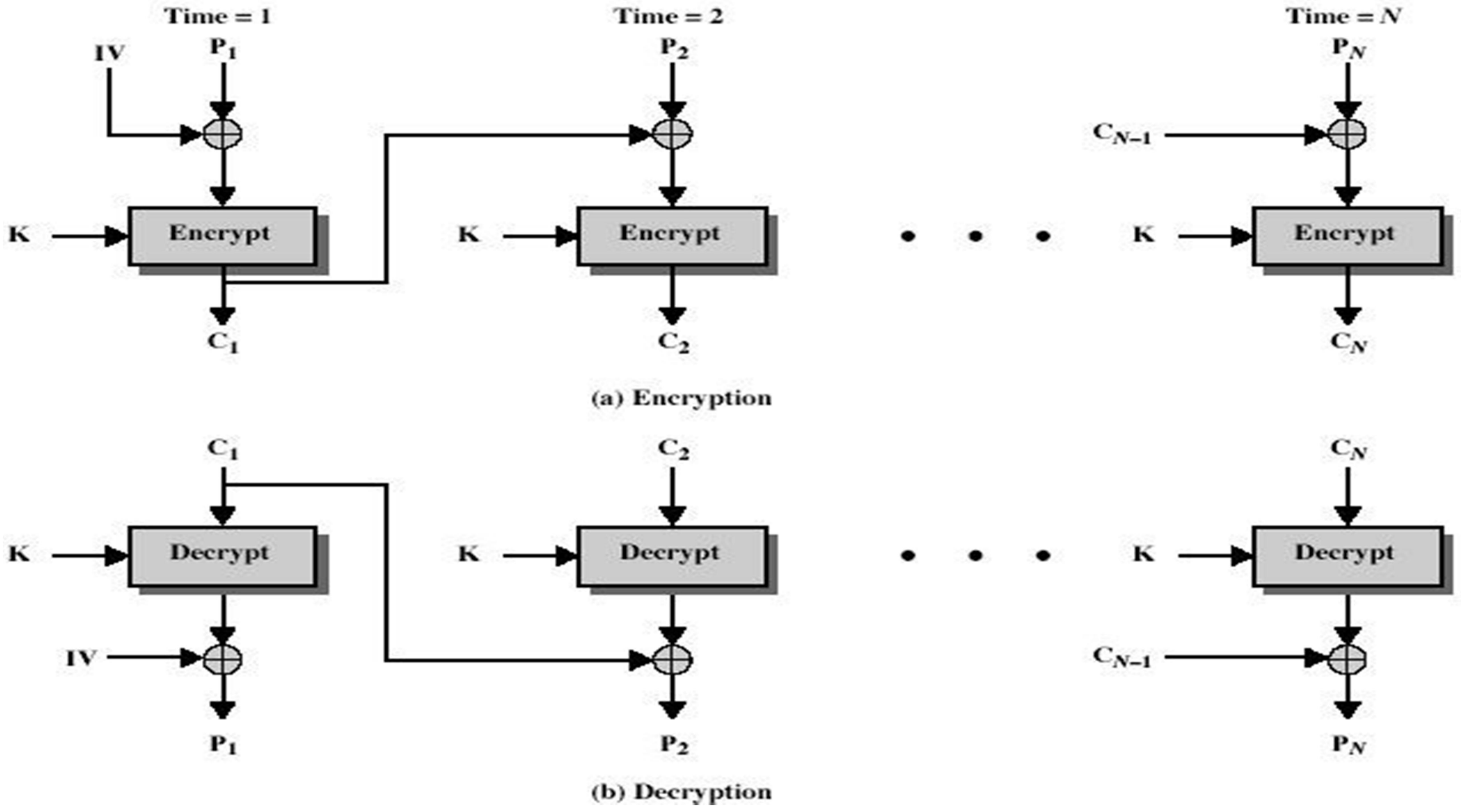
Cipher Block Chaining (CBC)

- message is broken into blocks
- but these are linked together in the encryption operation
- each previous cipher blocks is chained with current plaintext block, hence name
- use Initial Vector (IV) to start process

$C_i = DES_{K1}(P_i \text{ XoR } C_{i-1})$

$C_{-1} = IV$

Cipher Block Chaining (CBC)



Advantages and Limitations of CBC

- each ciphertext block depends on **all** message blocks
- thus a change in the message affects all ciphertext blocks after the change as well as the original block
- need **Initial Value** (IV) known to sender & receiver
  - however if IV is sent in the clear, an attacker can change bits of the first block, and change IV to compensate
  - hence either IV must be a fixed value (as in EFTPOS) or it must be sent encrypted in ECB mode before rest of message

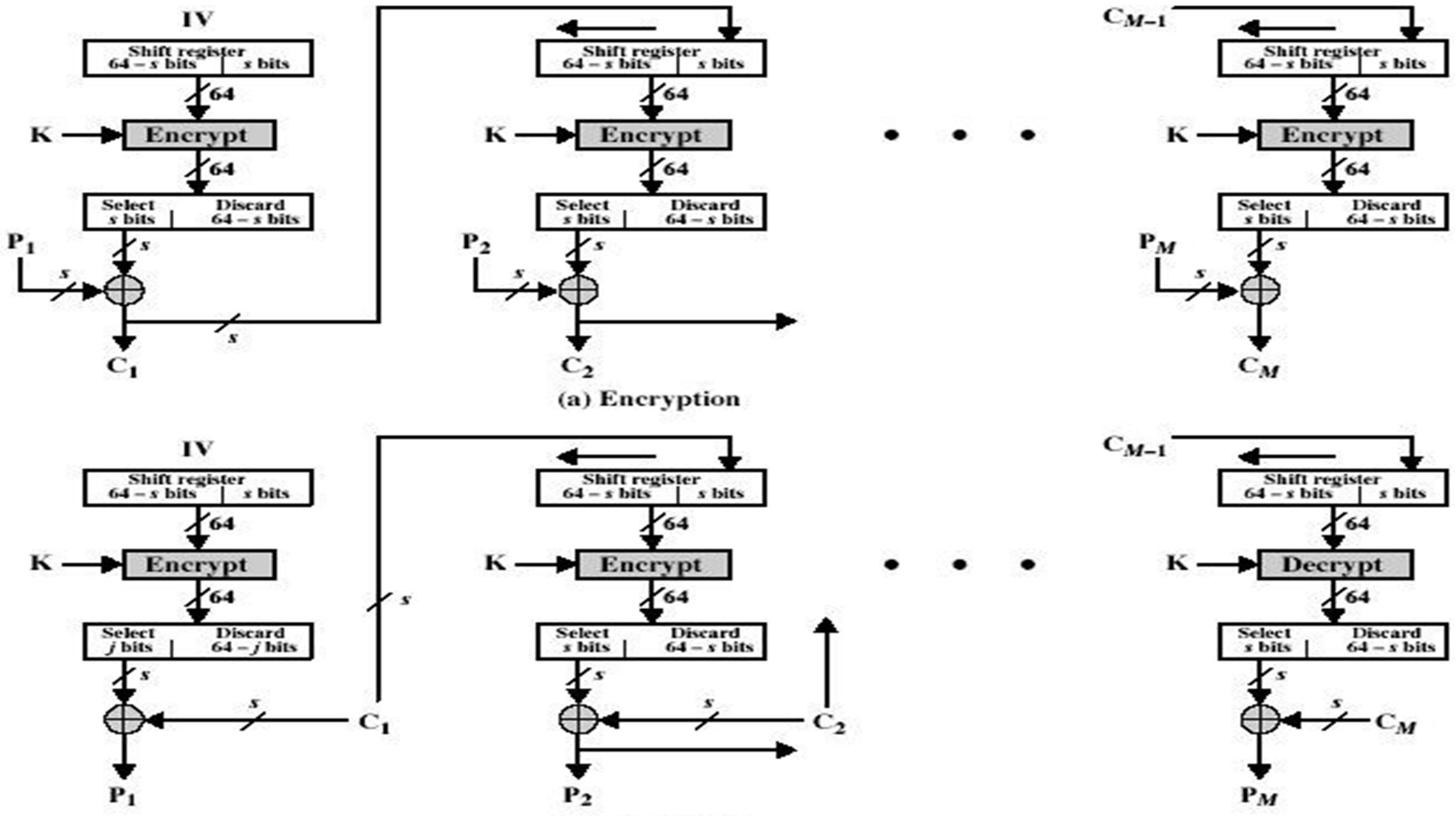
Cipher FeedBack (CFB)

- message is treated as a stream of bits
- added to the output of the block cipher
- result is feed back for next stage (hence name)
- standard allows any number of bit (1,8 or 64 or whatever) to be feed back
  - denoted CFB-1, CFB-8, CFB-64 etc
- is most efficient to use all 64 bits (CFB-64)

  $C_i = P_i \text{ XoR } DES_{K1}(C_{i-1})$

  $C_{-1} = IV$
- uses: stream data encryption, authentication

Cipher FeedBack (CFB)



IV
Shift register 64 – s bits | s bits
64
K
Encrypt
Decrypt
Select s bits | Discard 64 – s bits
Select j bits | Discard 64 – j bits
s
P₁
P₂
P_M
C₁
C₂
C_{M–1}
C_M

(a) Encryption

(b) Decryption

Advantages and Limitations of CFB

- appropriate when data arrives in bits/bytes
- most common stream mode
- note that the block cipher is used in **encryption** mode at **both** ends
- errors propagate for several blocks after the error
  - Must use over a reliable network channel

Output FeedBack (OFB)

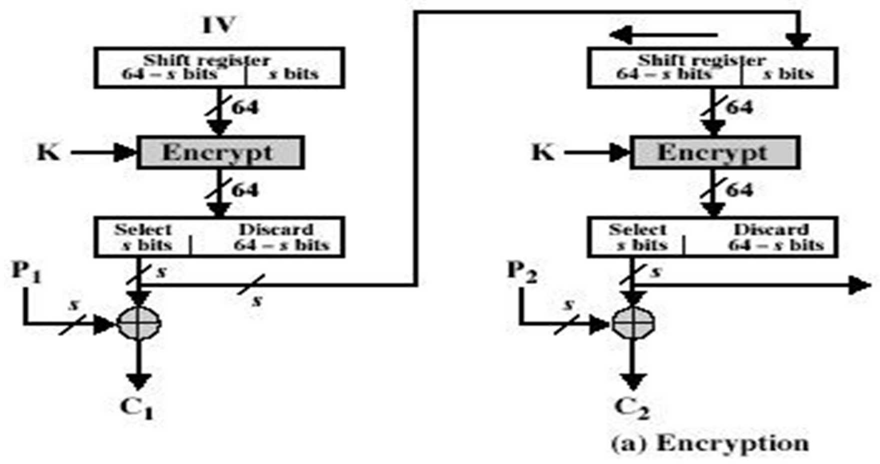
- message is treated as a stream of bits
- output of cipher is added to message
- output is then feed back (hence name)
- feedback is independent of message
- can be computed in advance

  $C_i = P_i \text{ XoR } o_i$

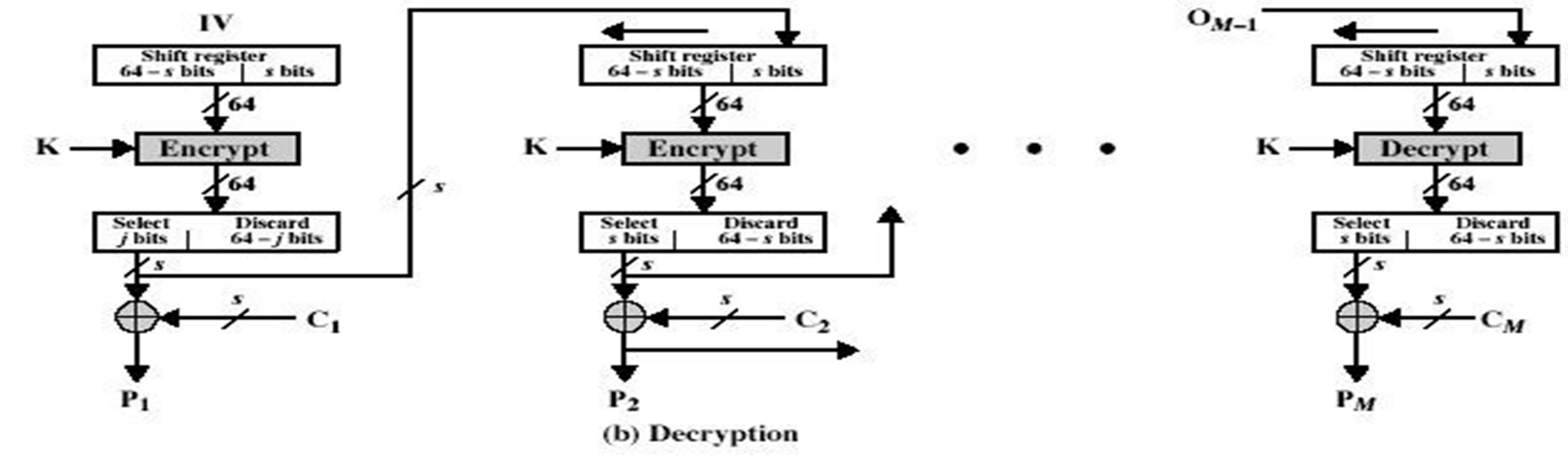
  $o_i = DES_{K1}(o_{i-1})$

  $o_{-1} = IV$
- uses: stream encryption over noisy channels

Output FeedBack (OFB)



(a) Encryption



(b) Decryption

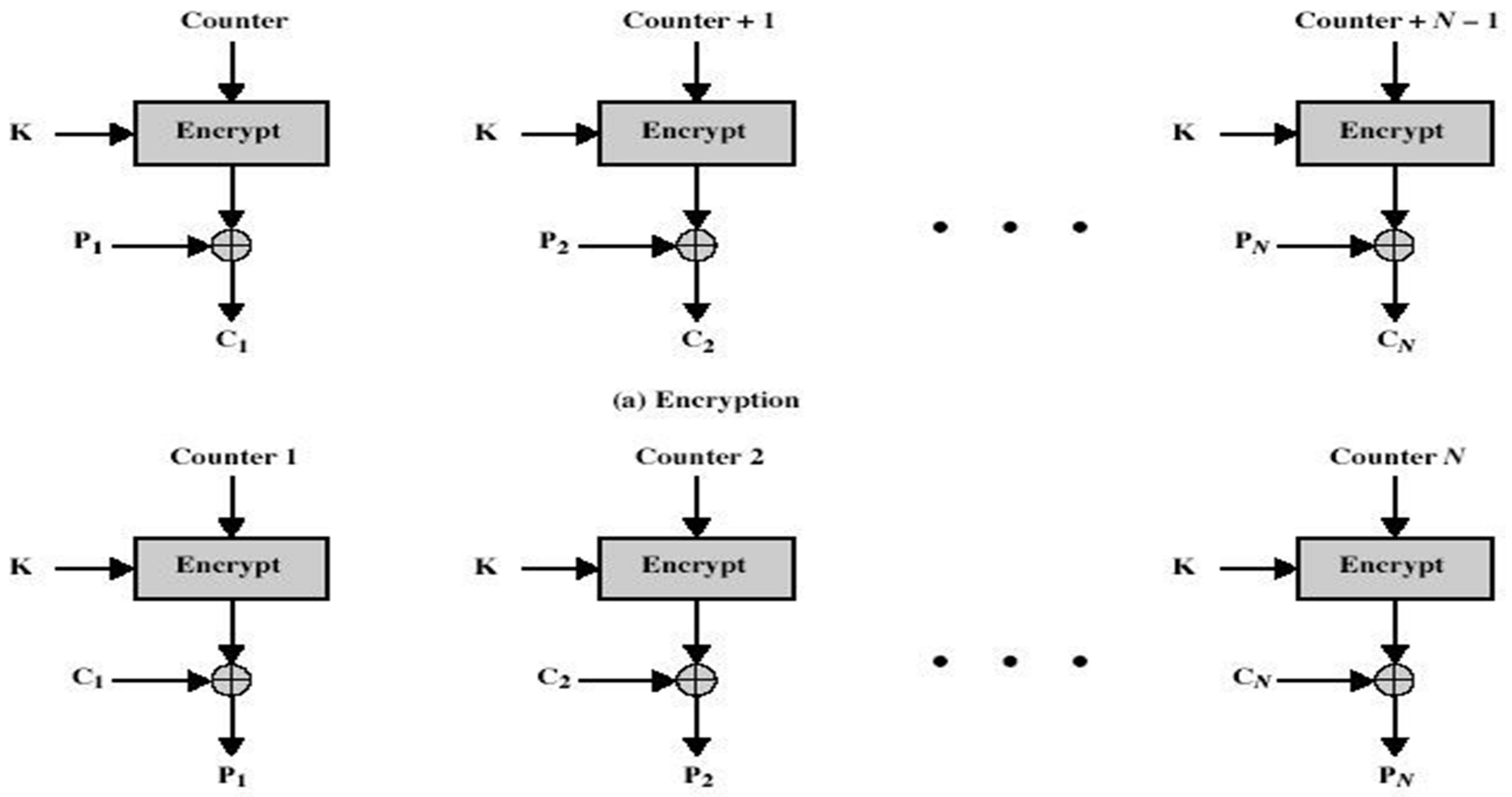
Counter (CTR)

- a “new” mode, though proposed early on
- encrypts counter value rather than any feedback value
- must have a different key & counter value for every plaintext block (never reused)

  $C_i = P_i \text{ XoR } o_i$

  $o_i = DES_{K1}(i)$
- uses: high-speed network encryptions

Counter (CTR)



(a) Encryption

(b) Decryption

Advantages and Limitations of CTR

- efficiency
  - can do parallel encryptions
  - in advance of need
  - good for bursty high speed links
- random access to encrypted data blocks
  - Do not have to decode from the beginning
- provable security (good as other modes)
- but must ensure never reuse key/counter

Euclid's GCD Algorithm

- an efficient way to find the GCD(a,b)
- uses theorem that:
  - GCD(a,b) = GCD(b, a mod b)
- **Euclid's Algorithm** to compute GCD(a,b):

```
– A=a,  B=b
– while  B>0
      • R = A mod B
      • A = B, B = R
– return  A
```

Polynomial Arithmetic

- can compute using polynomials
- se

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0 = \sum_{i=0}^{n} a_i x^i$$

  –
  – poly arithmetic with coefficients mod p
  – poly arithmetic with coefficients mod p and polynomials mod another polynomial M(x)
- Motivation: use polynomials to model Shift and XOR operations

Ordinary Polynomial Arithmetic

- add or subtract corresponding coefficients
- multiply all terms by each other
- eg
  - let $f(x) = x^3 + x^2 + 2$ and $g(x) = x^2 - x + 1$

    $f(x) + g(x) = x^3 + 2x^2 - x + 3$

    $f(x) - g(x) = x^3 + x + 1$

    $f(x) \text{ x } g(x) = x^5 + 3x^2 - 2x + 2$

AES Requirements

- private key symmetric block cipher
- 128-bit data, 128/192/256-bit keys
- stronger & faster than Triple-DES
- active life of 20-30 years (+ archival use)
- provide full specification & design details
- both C & Java implementations
- NIST have released all submissions & unclassified analyses

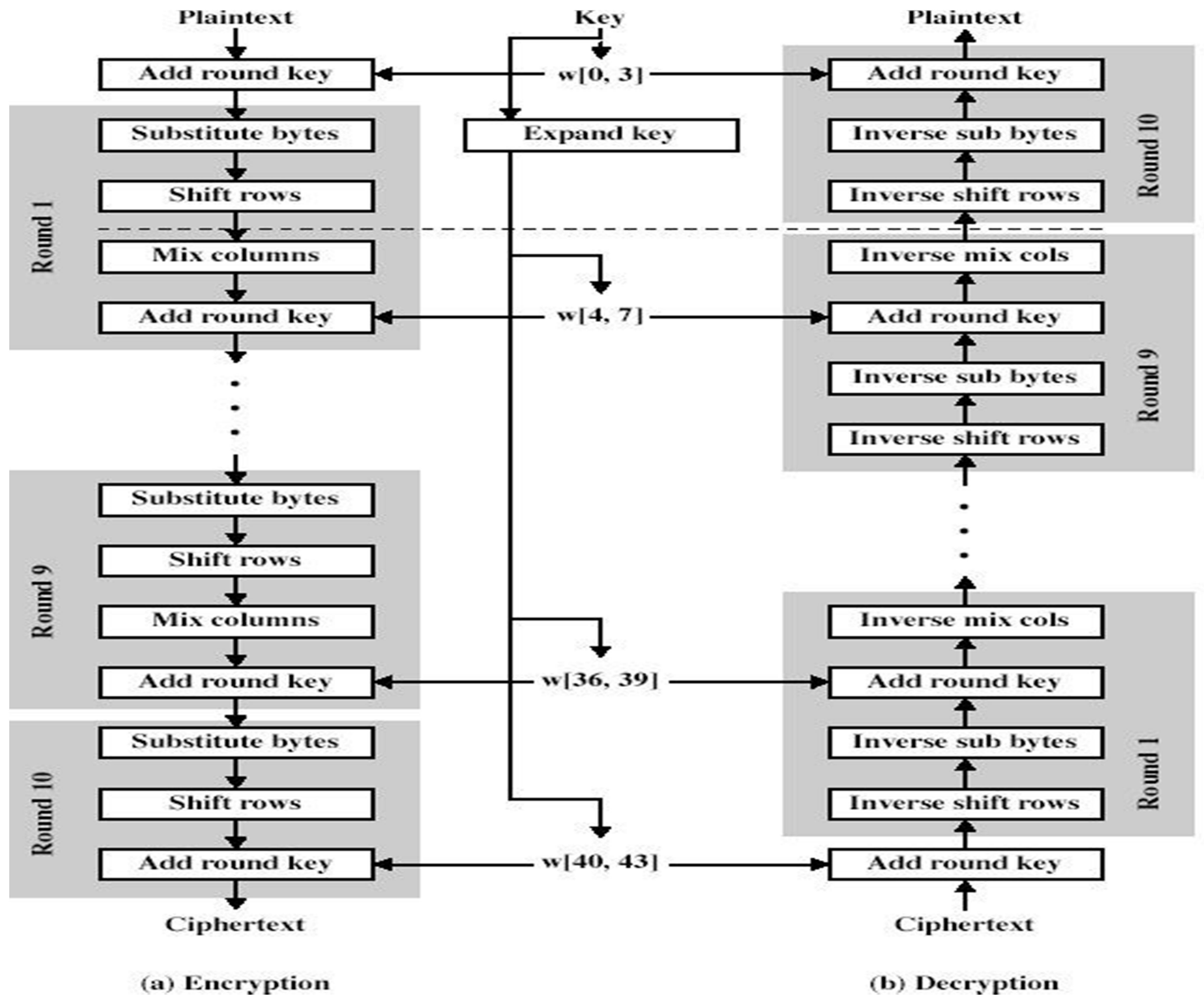
The AES Cipher

- designed by Rijmen-Daemen in Belgium
- has 128/192/256 bit keys, 128 bit data
- an **iterative** rather than **feistel** cipher
  - treats data in 4 groups of 4 bytes
  - operates an entire block in every round
  - rather than feistel (operate on halves at a time)
- designed to be:
  - resistant against known attacks
  - speed and code compactness on many CPUs
  - design simplicity

AES

- processes data as 4 groups of 4 bytes (state)
- has 9/11/13 rounds in which state undergoes:
  - byte substitution (1 S-box used on every byte)
  - shift rows (permute bytes between groups/columns)
  - mix columns (subs using matrix multiply of groups)
  - add round key (XOR state with key material)
- initial XOR key material & incomplete last round
- all operations can be combined into XOR and table lookups - hence very fast & efficient

Rijndael


Plaintext
Key
Plaintext
Add round key
w[0, 3]
Add round key
Substitute bytes
Expand key
Inverse sub bytes
Shift rows
Inverse shift rows
Round 1
Round 10
Mix columns
Inverse mix cols
Add round key
w[4, 7]
Add round key
Inverse sub bytes
Inverse shift rows
Round 9
Substitute bytes
Shift rows
Round 9
Mix columns
Inverse mix cols
Add round key
w[36, 39]
Add round key
Substitute bytes
Inverse sub bytes
Round 1
Shift rows
Inverse shift rows
Round 10
Add round key
w[40, 43]
Add round key
Ciphertext
Ciphertext
(a) Encryption
(b) Decryption

Byte Substitution

- a simple substitution of each byte
- uses one table of 16x16 bytes containing a permutation of all 256 8-bit values
- each byte of state is replaced by byte in row (left 4-bits) & column (right 4-bits)
  - eg. byte {95} is replaced by row 9 col 5 byte
  - which is the value {2A}
- S-box is constructed using a defined transformation of the values in GF(2^8)
- designed to be resistant to all known attacks

Shift Rows

- a circular byte shift in each row
  - 1st row is unchanged
  - 2nd row does 1 byte circular shift to left
  - 3rd row does 2 byte circular shift to left
  - 4th row does 3 byte circular shift to left
- decrypt does shifts to right
- since state is processed by columns, this step permutes bytes between the columns

Mix Columns

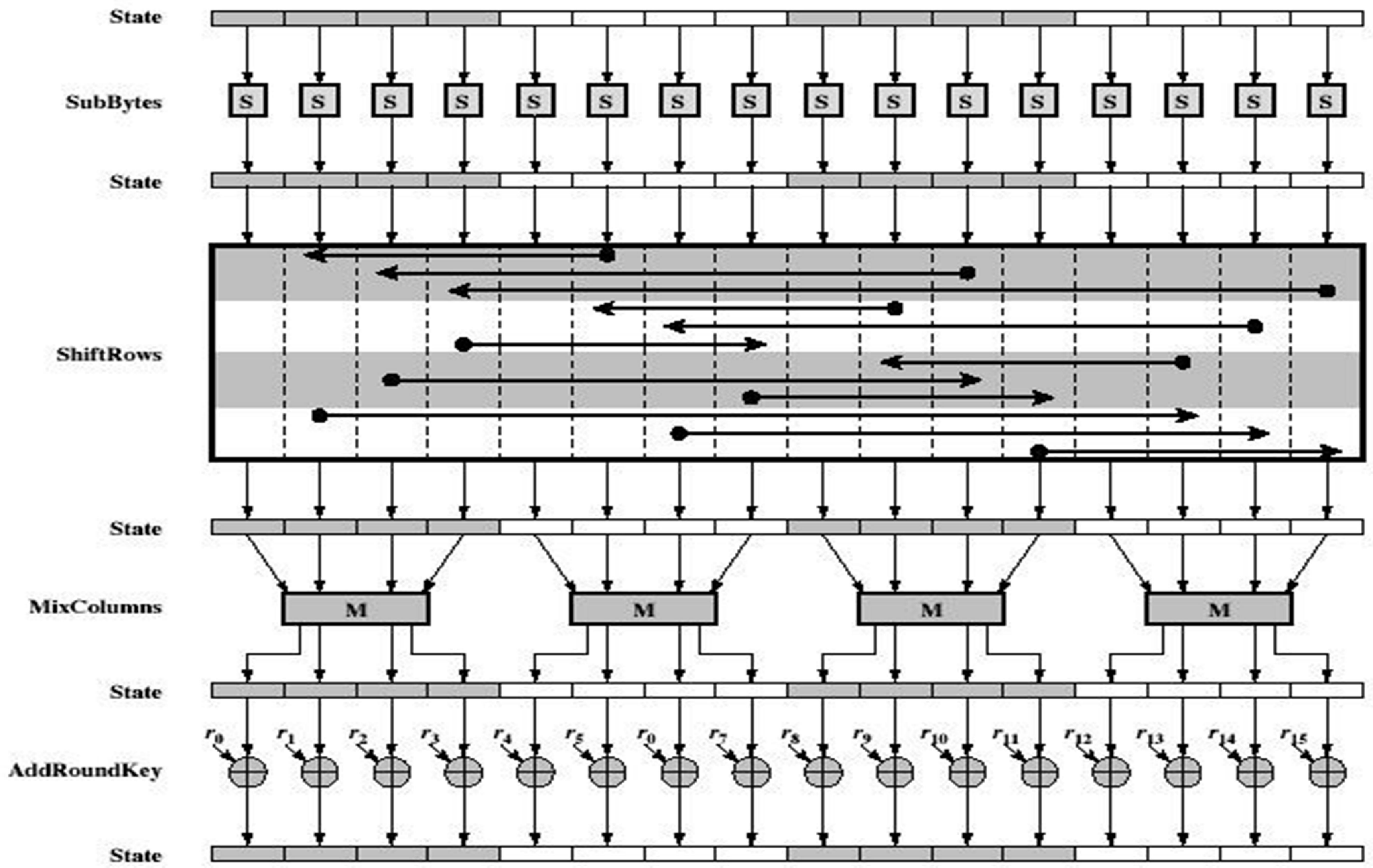
- each column is processed separately
- each byte is replaced by a value dependent on all 4 bytes in the column
- effectively a matrix multiplication in GF(2^8) using prime poly $m(x) = x^8+x^4+x^3+x+1$

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

Add Round Key

- XOR state with 128-bits of the round key
- again processed by column (though effectively a series of byte operations)
- inverse for decryption is identical since XOR is own inverse, just with correct round key
- designed to be as simple as possible

AES Round


State
SubBytes
S
State
ShiftRows
State
MixColumns
M
State
AddRoundKey
r_0 r_1 r_2 r_3 r_4 r_5 r_0 r_7 r_8 r_9 r_{10} r_{11} r_{12} r_{13} r_{14} r_{15}
State

AES Key Expansion

- takes 128-bit (16-byte) key and expands into array of 44/52/60 32-bit words
- start by copying key into first 4 words
- then loop creating words that depend on values in previous & 4 places back
  - in 3 of 4 cases just XOR these together
  - every 4th has S-box + rotate + XOR constant of previous before XOR together
- designed to resist known attacks

AES Decryption

- AES decryption is not identical to encryption since steps done in reverse
- but can define an equivalent inverse cipher with steps as for encryption
  - but using inverses of each step
  - with a different key schedule
- works since result is unchanged when
  - swap byte substitution & shift rows
  - swap mix columns & add (tweaked) round key

Implementation Aspects

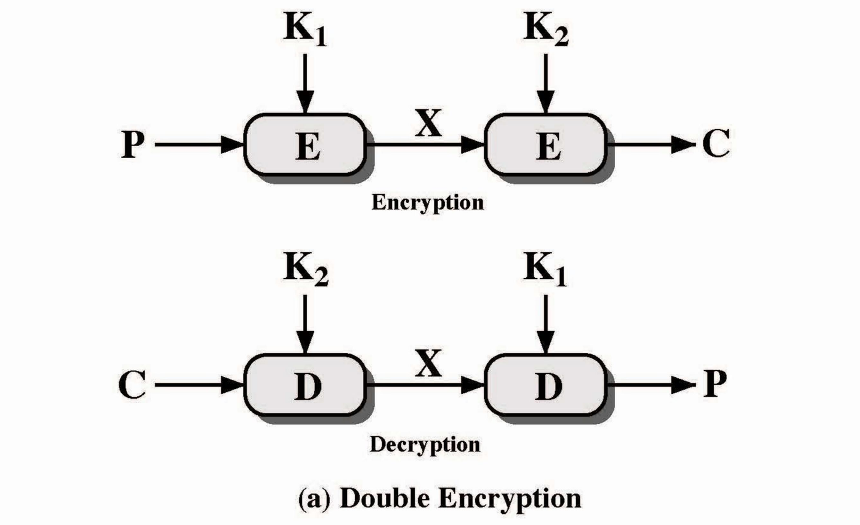
- can efficiently implement on 8-bit CPU
  - byte substitution works on bytes using a table of 256 entries
  - shift rows is simple byte shifting
  - add round key works on byte XORs
  - mix columns requires matrix multiply in GF(2^8) which works on byte values, can be simplified to use a table lookup

Implementation Aspects

- can efficiently implement on 32-bit CPU
  - redefine steps to use 32-bit words
  - can pre-compute 4 tables of 256-words
  - then each column in each round can be computed using 4 table lookups + 4 XORs
  - at a cost of 16Kb to store tables
- designers believe this very efficient implementation was a key factor in its selection as the AES cipher

Triple DES

- A replacement for DES was needed
  - theoretical attacks that can break it
  - demonstrated exhaustive key search attacks
- AES is a new cipher alternative
- Before AES alternative
  - use multiple encryptions with DES
- Triple-DES is the chosen form



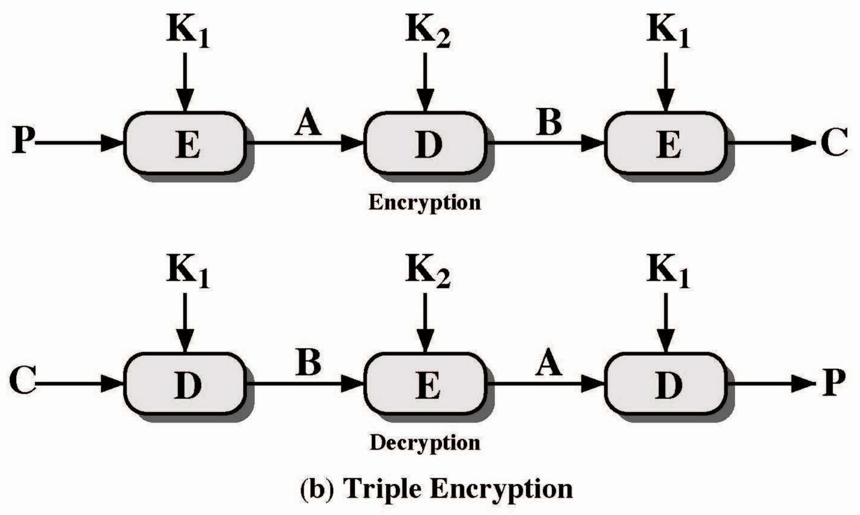


Figure 6.1 Multiple Encryption

Why Triple-DES?

- why not Double-DES?
  - NOT same as some other single-DES use, but have
- meet-in-the-middle attack
  - works whenever use a cipher twice
  - since X = $E_{K1}[P]$ = $D_{K2}[C]$
  - attack by encrypting P with all keys and store
  - then decrypt C with keys and match X value
  - can show takes $o(2^{56})$ steps

Triple-DES with Two-Keys

- hence must use 3 encryptions
  - would seem to need 3 distinct keys
  - Key of 56 X 3 = 168 bits seems too long
- but can use 2 keys with E-D-E sequence
  - $C = E_{K1}[D_{K2}[E_{K1}[P]]]$
  - No cryptographic significance to the use of D in the second step
- standardized in ANSI X9.17 & ISO8732
- no current known practical attacks
  - some are now adopting Triple-DES with three keys for greater security

Triple-DES with Three-Keys

- although are no practical attacks on two-key Triple-DES have some indications
- can use Triple-DES with Three-Keys to avoid even these
  - $C = E_{K3}[D_{K2}[E_{K1}[P]]]$
- has been adopted by some Internet applications

Blowfish

- a symmetric block cipher designed by Bruce Schneier in 1993/94
- characteristics
  - fast implementation on 32-bit CPUs, 18 clock cycles per byte
  - compact in use of memory, less than 5KB
  - simple structure for analysis/implementation
  - variable security by varying key size
    - Allows tuning for speed/security tradeoff

Blowfish Key Schedule

- uses a 32 to 448 bit key
- used to generate
  - 18 32-bit subkeys stored in P-array: P1 to P18
  - S-boxes stored in $S_{i,j,}$
    - i=1..4
    - j=0..255

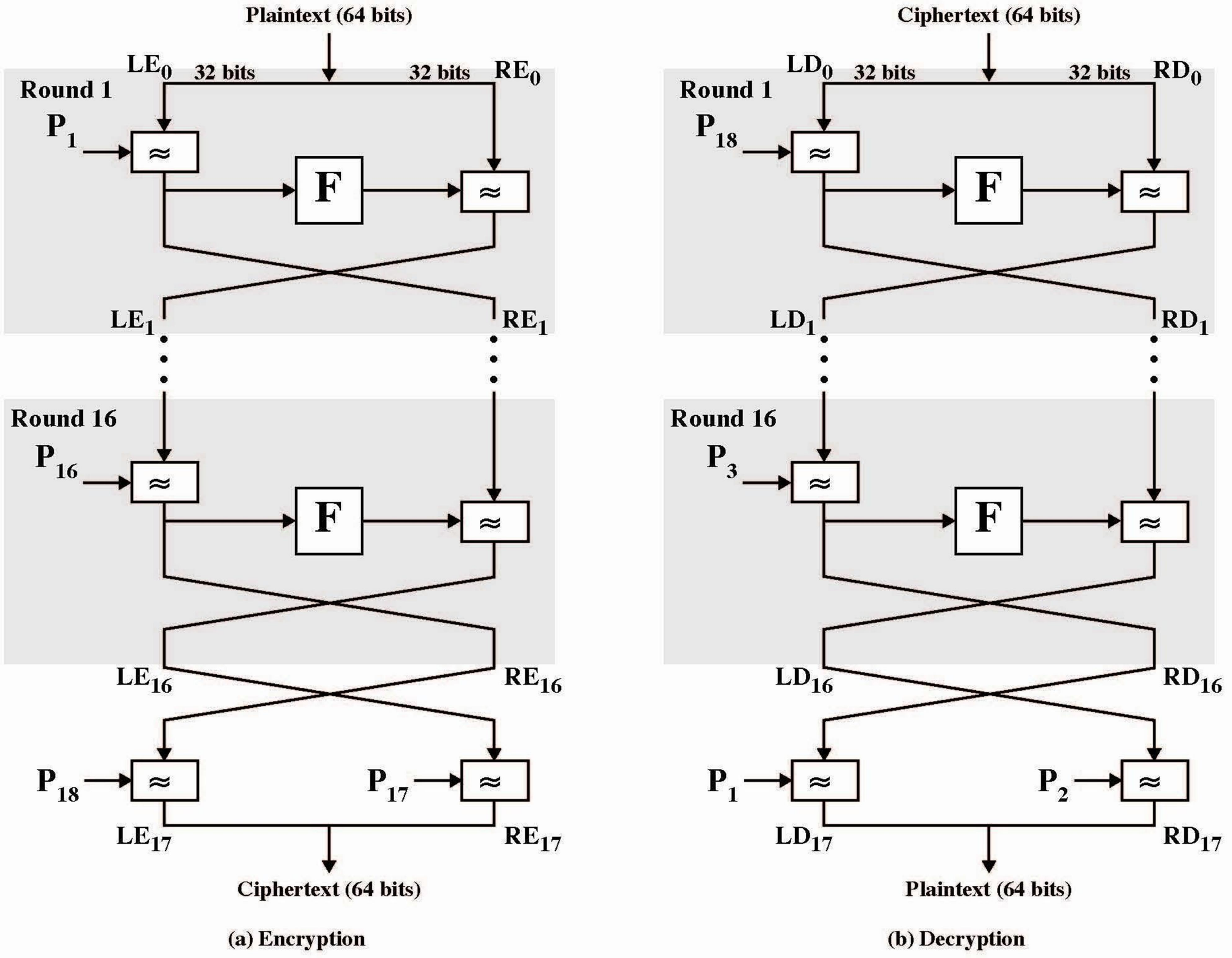


Figure 6.3 Blowfish Encryption and Decryption

Blowfish Encryption

- uses two primitives: addition & XOR
- data is divided into two 32-bit halves L_0 & R_0

```
for $i$ = 1 to 16 do
    $R_i$ = $L_{i-1}$ XoR $P_i$;
    $L_i$ = F[$R_i$] XoR $R_{i-1}$;
$L_{17}$ = $R_{16}$ XoR $P_{18}$;
$R_{17}$ = $L_{16}$ XoR $i_{17}$;
```

- where

  F[a,b,c,d] = $((S_{1,a} + S_{2,b})$ XoR $S_{3,c}) + S_{4,a}$

  Break 32-bit R_i into (a,b,c,d)

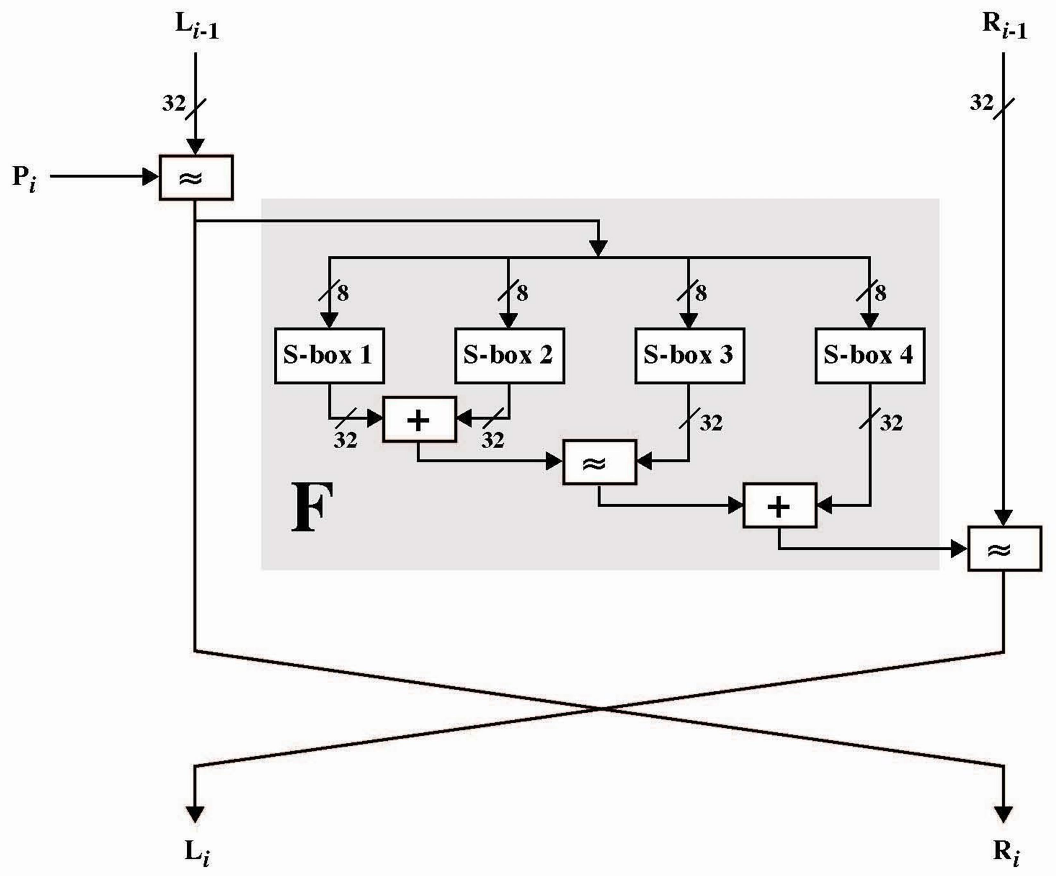


Figure 6.4 Detail of Single Blowfish Round

RC5

- can vary key size / input data size / #rounds
- very clean and simple design
- easy implementation on various CPUs
- yet still regarded as secure
  - Vary parameters to achieve tradeoffs

RC5 Ciphers

- RC5 is a family of ciphers RC5-w/r/b
  - w = word size in bits (16/32/64) data=2w
  - r = number of rounds (0..255)
  - b = number of bytes in key (0..255)
- nominal version is RC5-32/12/16
  - ie 32-bit words so encrypts 64-bit data blocks
  - using 12 rounds
  - with 16 bytes (128-bit) secret key

RC5 Key Expansion

- RC5 uses 2r+2 subkey words (w-bits)
  - Two subkeys for each round
  - 2 subkeys for additional operations
- subkeys are stored in array S[i], i=0..t-1
- Key expansion: fill in pseudo-random bits to the original key K
- Certain amount of *one-wayness*
  - Difficult to determine K from S

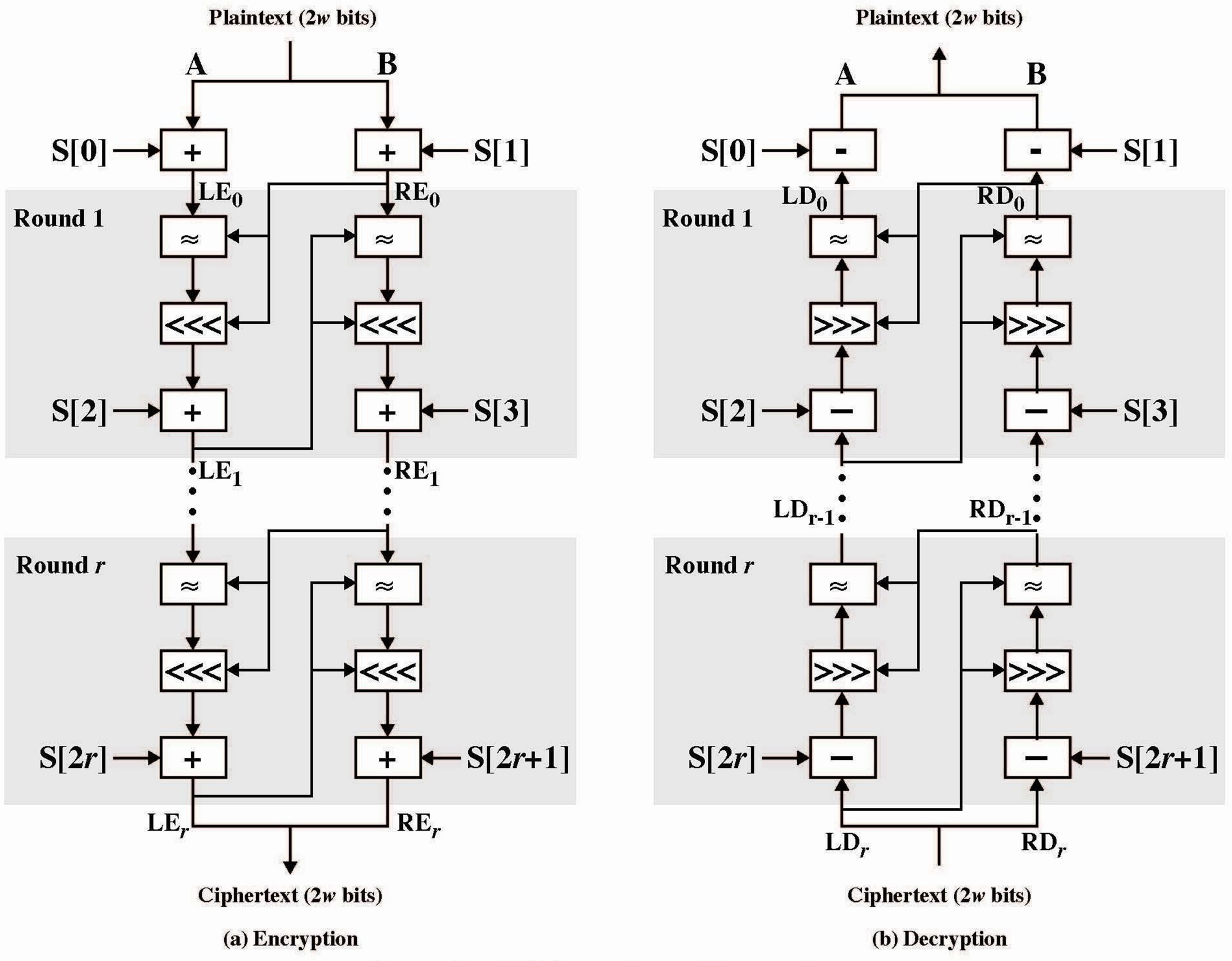


Figure 6.6 RC5 Encryption and Decryption

RC5 Encryption

- split input into two halves A & B

```
$L_0 = A + S[0];$
$R_0 = B + S[1];$
for $i$ = 1 to $r$ do
    $L_i = ((L_{i-1} \text{ XOR } R_{i-1}) <<< R_{i-1}) + S[2 \times i];$
    $R_i = ((R_{i-1} \text{ XOR } L_i) <<< L_i) + S[2 \times i + 1];$
```

- each round is like 2 DES rounds
- note rotation is main source of non-linearity
- need reasonable number of rounds (eg 12-16)
- Striking features: simplicity, data-dependent rotations

RC5 Modes

- RFC2040 defines 4 modes used by RC5
  - RC5 Block Cipher, is ECB mode
  - RC5-CBC, input length is a multiples of 2w
  - RC5-CBC-PAD, any length CBC with padding
    - Output can be longer than input
  - RC5-CTS, CBC with padding
    - Output has same length than input

Block Cipher Characteristics

- features seen in modern block ciphers are:
  - variable key length / block size / no rounds
  - mixed operators
    - data/key dependent rotation
    - key dependent S-boxes
  - more complex key scheduling
    - Lengthy key generation, simple encryption rounds
  - operation of full data in each round

Stream Ciphers

- process the message bit by bit (as a stream)
- typically have a (pseudo) random **key stream**
- combined (XOR) with plaintext bit by bit
- randomness of **key stream** completely destroys any statistically properties in the message
  - $C_i = M_i$ XoR StreamKey$_i$
- what could be simpler!!!!
- but must never reuse key stream
  - otherwise can remove effect and recover messages

Block/Stream Ciphers

- Stream ciphers
  - For applications that require encryt/decryt of a stream of data
  - Examples: data communication channel, brower/web link
- Block ciphers
  - For applications dealing with blocks of data
  - Examples: file transfer, e-mail, database
- Either type can be used in virtually any application

Stream Cipher Properties

- some design considerations are:
  - long period with no repetitions
  - statistically random
  - Highly nonlinear correlation

RC4

- variable key size, byte-oriented stream cipher
- widely used (web SSL/TLS between browser and server, wireless WEP)
- key forms random permutation of a 8-bit string
- uses that permutation to scramble input info processed a byte at a time

RC4 Security

- claimed secure against known attacks
  - have some analyses, none practical
- result is very non-linear
- since RC4 is a stream cipher, must **never reuse a key**

Placement of Security Devices

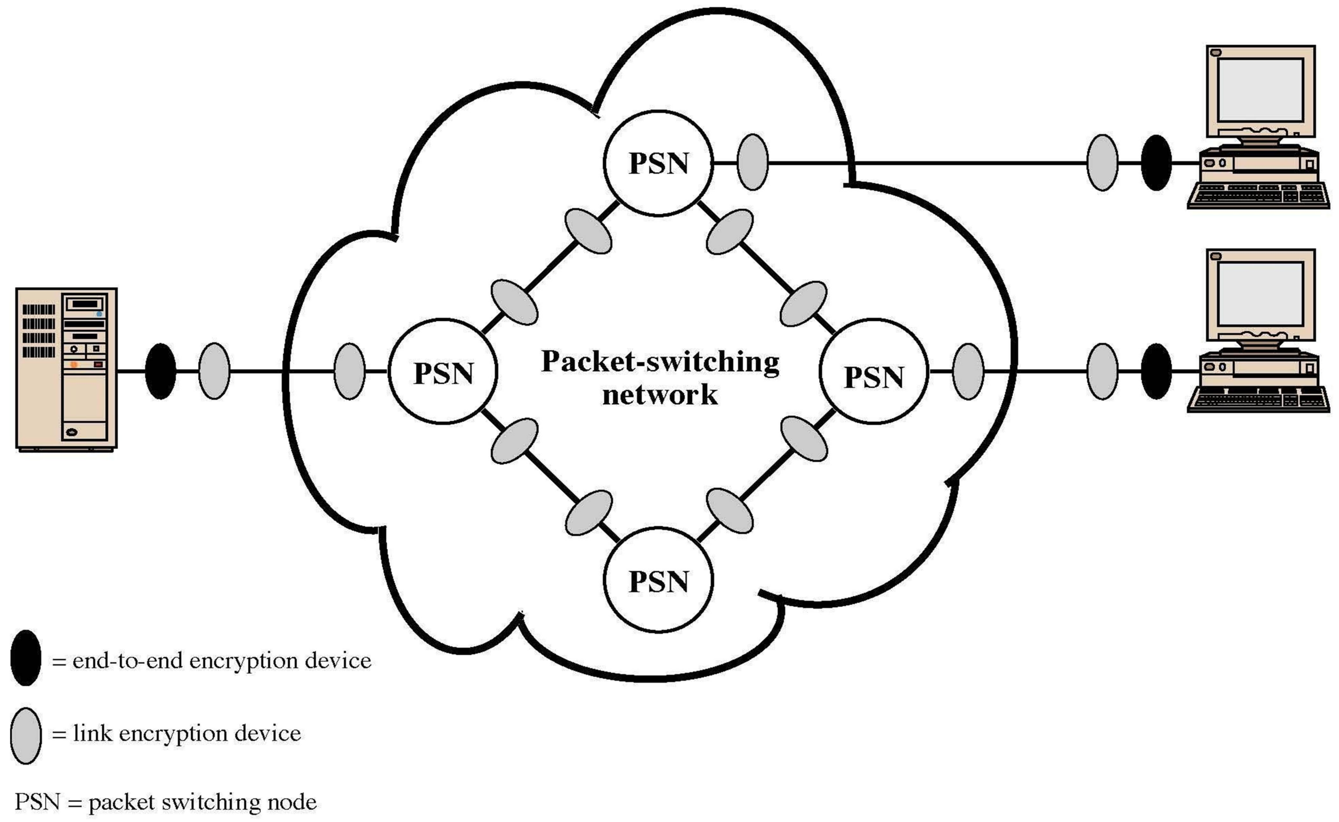


Figure 7.2 Encryption Across a Packet-Switching Network

Two major placement alternatives

- **link encryption**
  - encryption occurs independently on every link
  - implies must decrypt traffic between links
  - One key per (node, node) pair
  - Message exposed in nodes
  - Transparent to user, done in hardware
- **end-to-end encryption**
  - encryption occurs between original source and final destination
  - One key per user pair
  - Message encrypted in nodes

Traffic Analysis

- when using end-to-end encryption must leave headers in clear
  - so network can correctly route information
- hence although contents protected, traffic pattern flows are not

Key Distribution

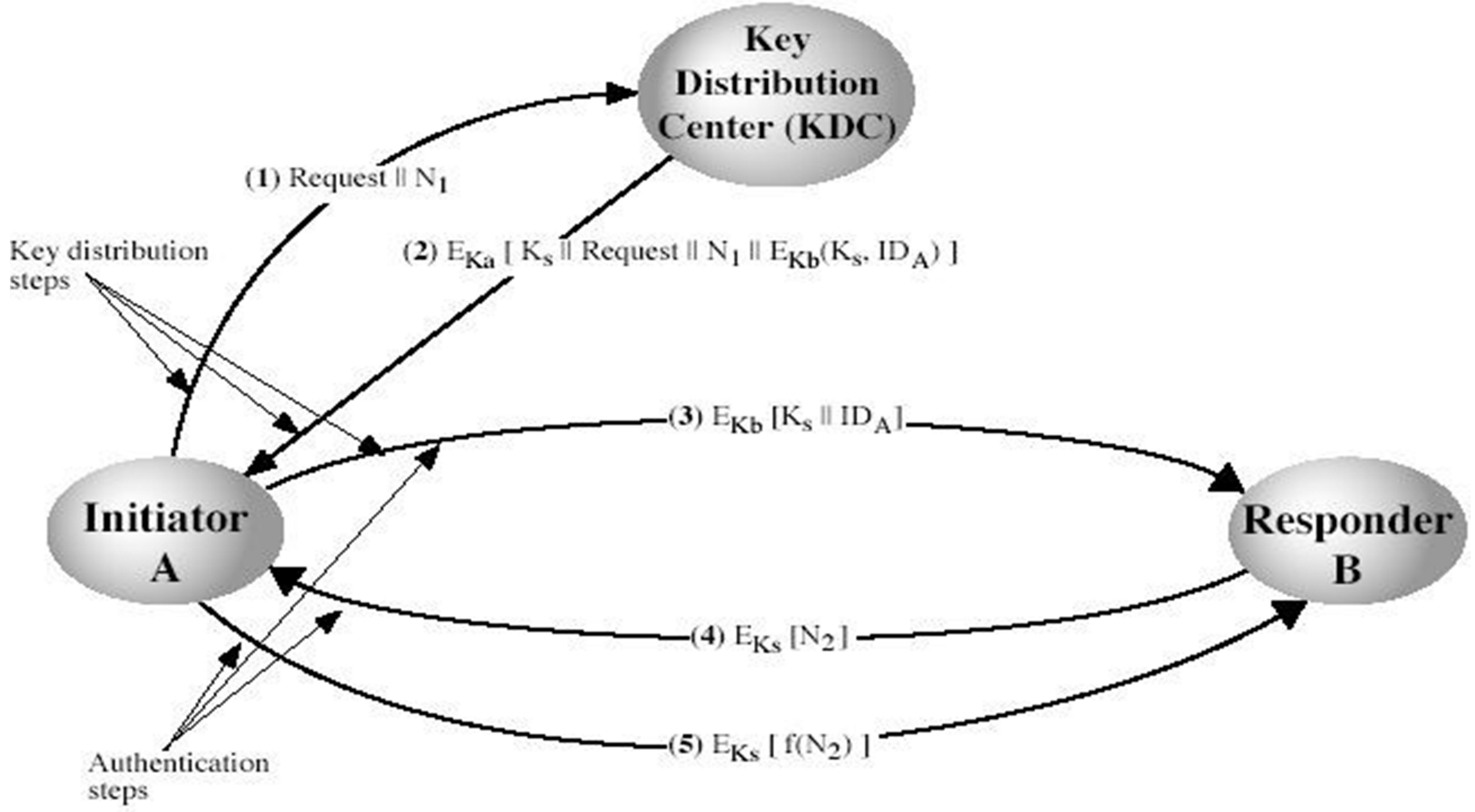
- symmetric schemes require both parties to share a common secret key
- issue is how to securely distribute this key
- often secure system failure due to a break in the key distribution scheme

Key Distribution

- given parties A and B have various **key distribution** alternatives:
  1. A can select key and physically deliver to B
  2. third party can select & deliver key to A & B
  3. if A & B have communicated previously can use previous key to encrypt a new key
  4. if A & B have secure communications with a third party C, C can relay key between A & B

As number of parties grow, some variant of 4 is only practical solution.

Key Distribution Scenario



Key Distribution Center (KDC)
(1) Request || N_1
Key distribution steps
(2) E_{Ka} [K_s || Request || N_1 || $E_{Kb}(K_s, ID_A)$]
(3) E_{Kb} [K_s || ID_A]
Initiator A
Responder B
(4) E_{Ks} [N_2]
(5) E_{Ks} [$f(N_2)$]
Authentication steps

Random Numbers

- many uses of **random numbers** in cryptography
  - Ns in authentication protocols to prevent replay
  - session keys
  - public key generation
  - keystream for a one-time pad
- in all cases its critical that these values be
  - statistically random
    - with uniform distribution, independent
  - unpredictable cannot infer future sequence on previous

Using Block Ciphers as Stream Ciphers

- can use block cipher to generate numbers
- use Counter Mode

  $X_i = E_{Km}[i]$
- use Output Feedback Mode

  $X_i = E_{Km}[X_{i-1}]$
- ANSI standard, uses output feedback 3-DES

Private-Key Cryptography

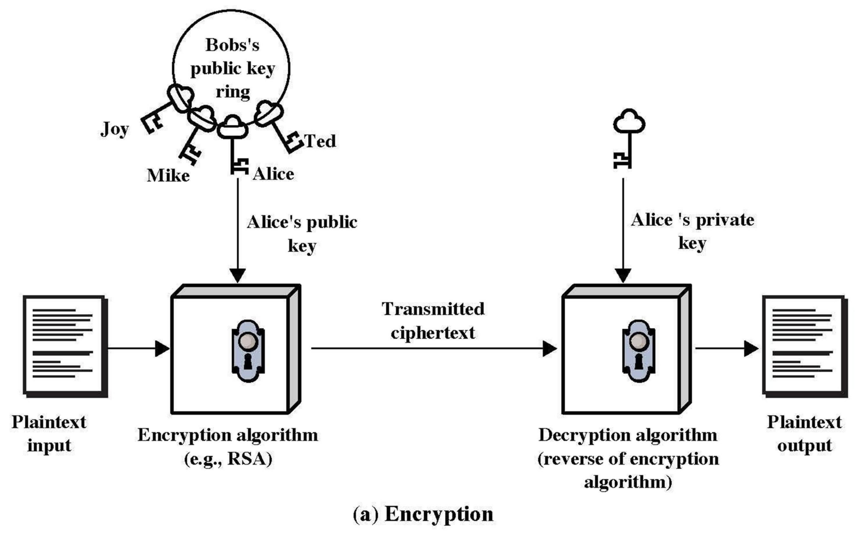
- traditional **private/secret/single key** cryptography uses **one** key
- shared by both sender and receiver
- if this key is disclosed, communications are compromised
- also is **symmetric**, parties are equal
- hence does not protect sender from receiver forging a message & claiming is sent by sender

Public-Key Cryptography

- probably most significant advance in the 3000 year history of cryptography
- uses **two** keys – a public & a private key
  - Anyone knowing the public key can encrypt messages or verify signatures
  - **But cannot** decrypt messages or create signatures
- **asymmetric** since parties are **not** equal
- complements **rather than** replaces private key crypto

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



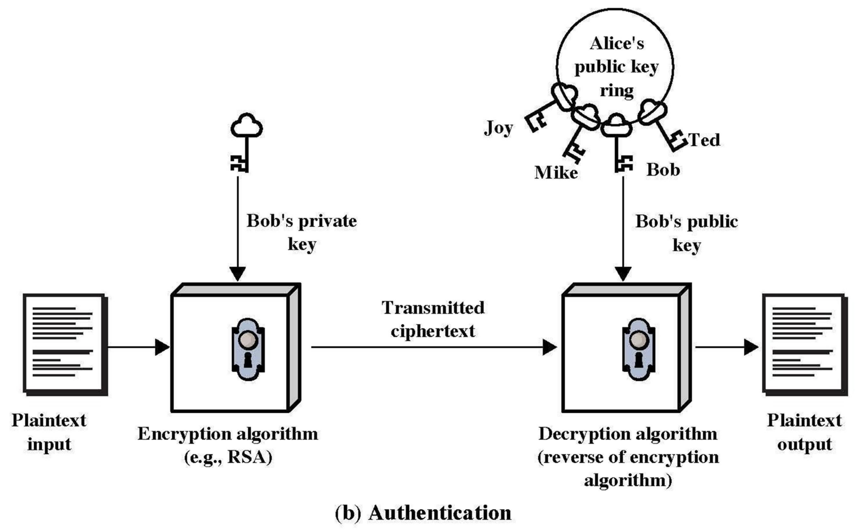


Figure 9.1 Public-Key Cryptography

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
    - No need for secure key delivery
    - No one else needs to know your private key
  - **digital signatures** – how to verify a message comes intact from the claimed sender

Public-Key Characteristics

- Public-Key algorithms rely on two keys with the characteristics that it is:
  - computationally infeasible to find decryption key knowing only algorithm & encryption key
  - computationally easy to en/decrypt messages when the relevant (en/decrypt) key is known
  - Oneway-ness is desirable: exp/log, mul/fac
  - either of the two related keys can be used for encryption, with the other used for decryption (in some schemes)

Public-Key Cryptosystems: Secrecy

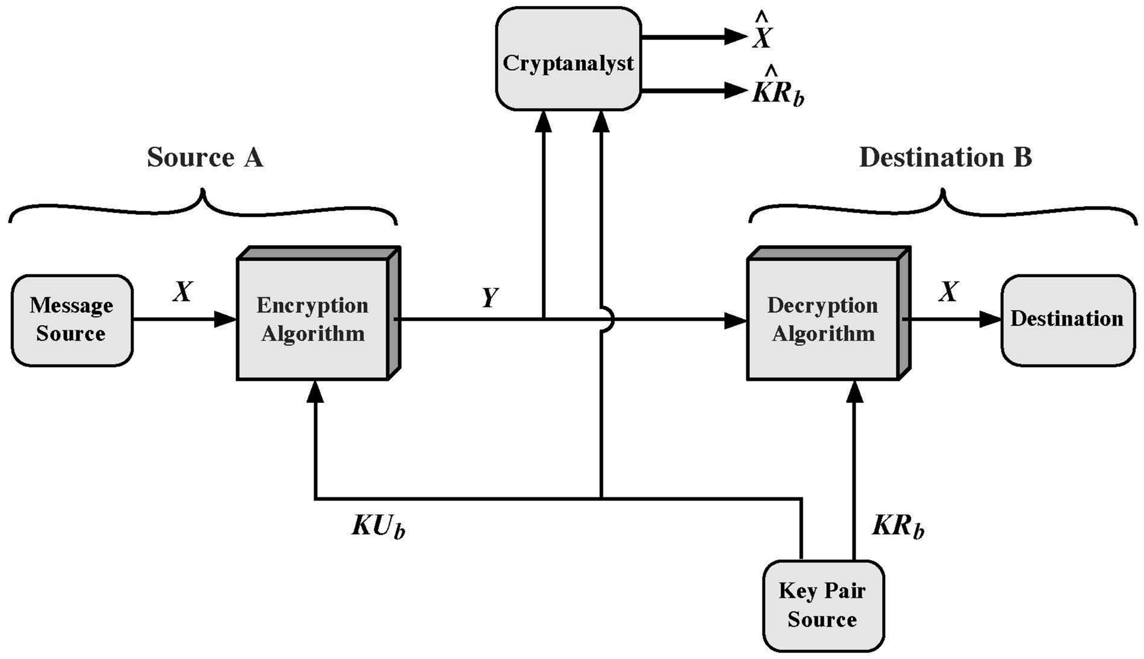


Figure 9.2 Public-Key Cryptosystem: Secrecy

Public-Key Cryptosystems: Authentication

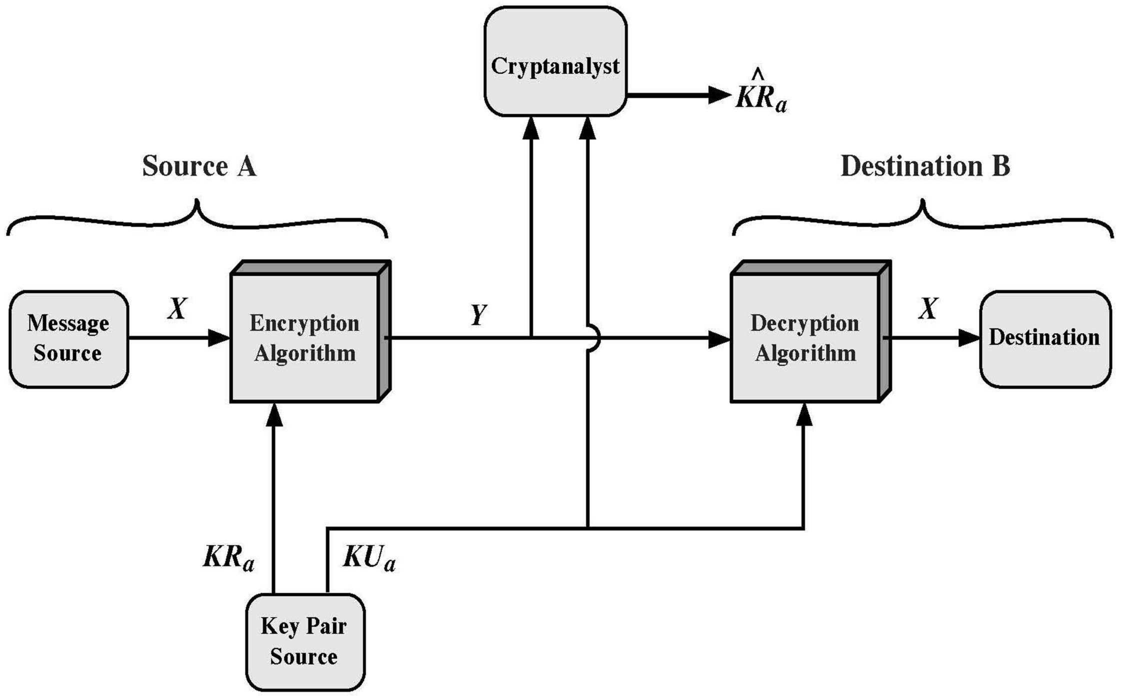


Figure 9.3 Public-Key Cryptosystem: Authentication

Public-Key Cryptosystems: Secrecy and Authentication

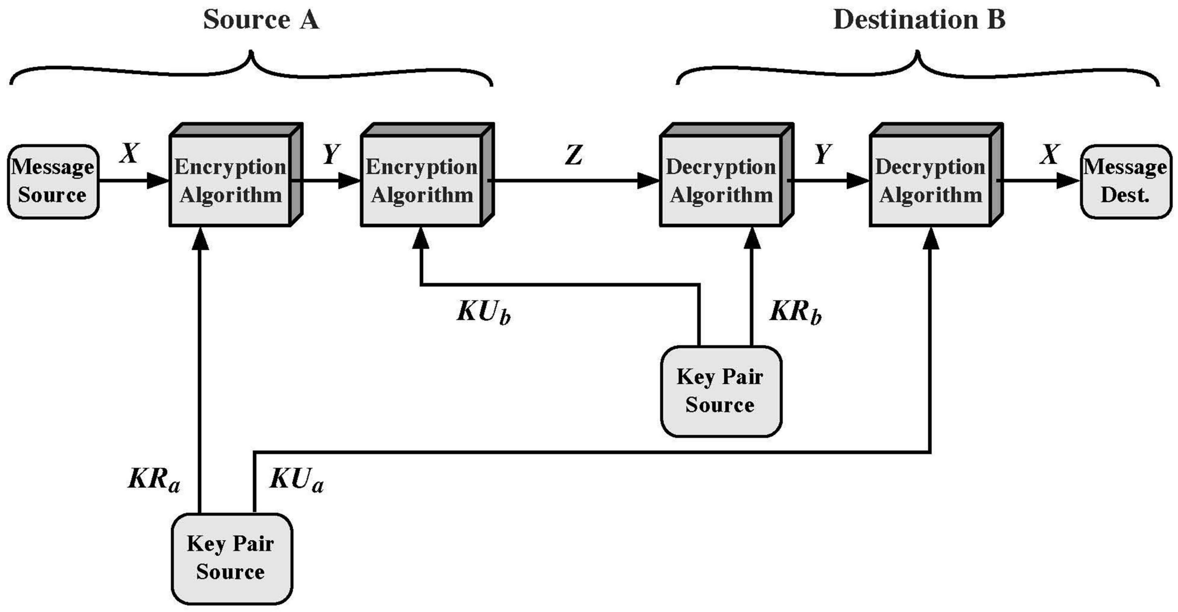


Figure 9.4 Public-Key Cryptosystem: Secrecy and Authentication

Public-Key Applications

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

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
- requires the use of **very large numbers**
- hence is **slow** compared to private key schemes

RSA

- by Rivest, Shamir & Adleman of MIT in 1977
- best known & widely used public-key scheme
- based on exponentiation of integers in a finite (Galois) field
  - Defined over integers modulo a prime
  - exponentiation takes $O((\log n)^3)$ operations (easy)
- uses large integers (eg. 1024 bits)
- security due to cost of factoring large numbers
  - factorization takes $O(e^{\log n \log \log n})$ operations (hard)

RSA Key Setup

- each user generates a public/private key pair by:
  1. selecting two large primes at random - p, q (secret)
  2. computing their system modulus $N=p.q$ (public)
     - note $ø(N)=(p-1)(q-1)$ (secret)
  3. selecting at random the encryption key e (public)
     - where $1<e<ø(N)$, $\gcd(e,ø(N))=1$
  4. solve following equation to find decryption key d (secret)
     - $e.d=1 \mod ø(N)$ and $0 \le d \le N$
     - Use the extended Euclid's algorithm to find the multiplicative inverse of e (mod ø(N))
- publish their public encryption key: KU={e,N}
- keep secret private decryption key: KR={d,p,q}

Block size of RSA

- Each block is represented as an integer number
- Each block has a value M less than N
- The block size is <= $\log_2(N)$ bits
- If the block size is k bits then

  $2^k <= N <= 2^{K+1}$

RSA Use

- to encrypt a message M the sender:
  - obtains **public key** of recipient KU={e,N}
  - computes: $C=M^e \bmod N$, where $0 \le M < N$
- to decrypt the ciphertext C the owner:
  - uses their private key KR={d,p,q}
  - computes: $M=C^d \bmod N$
- note that the message M must be smaller than the modulus N (block if needed)

Why RSA Works

- because of Euler's Theorem:
- $a^{ø(n)} \bmod N = 1$
  - where gcd(a,N)=1
- in RSA have:
  - N=p.q
  - ø(N)=(p-1)(q-1)
  - carefully chosen e & d to be inverses mod ø(N)
  - hence e.d=1+k.ø(N) for some k
- Two cases:
  - 1. gcd(M, N) = 1
  - 2. gcd(M, N) > 1, see equation (8.6) in P.243

RSA Example

1. Select primes: p=17 & q=11
2. Compute $n = pq$ =17×11=187
3. Compute ø(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}

RSA Example cont

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

  $C = 88^7 \bmod 187 = 11$
- decryption:

  $M = 11^{23} \bmod 187 = 88$

Exponentiation

```
$c \leftarrow 0$; $d \leftarrow 1$
for i $\leftarrow$ k downto 0
    do  $c \leftarrow 2 \times c$
        $d \leftarrow (d \times d) \bmod n$
        if  $b_i = 1$
            then  $c \leftarrow c + 1$
                  $d \leftarrow (d \times a) \bmod n$
return d
```

RSA Key Generation

- users of RSA must:
  - determine two primes at random - p, q
  - select either e or d and compute the other
- primes p,q must not be easily derived from modulus N=p.q
  - means must be sufficiently large
  - typically guess and use probabilistic test
- exponents e, d are inverses, so use Inverse algorithm to compute the other

RSA Security

- three 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 of decryption)

Factoring Problem

- mathematical approach takes 3 forms:
  - factor N=p.q, hence find ø(N) and then d
  - determine ø(N) directly and find d
  - find d directly
- currently believe all equivalent to factoring
  - have seen slow improvements over the years
    - as of Aug-99 best is 130 decimal digits (512) bit with GNFS
  - biggest improvement comes from improved algorithm
    - cf "Quadratic Sieve" to "Generalized Number Field Sieve"
  - barring dramatic breakthrough 1024+ bit RSA secure
    - ensure p, q of similar size and matching other constraints

Timing Attacks

- developed in mid-1990's
- exploit timing variations in operations
  - infer bits of d based on time taken
- countermeasures
  - use constant exponentiation time
  - add random delays
  - blind values used in calculations
    - $C' = (Mr)^e$, $M' = (C')^d$, $M=M'r^{-1}$

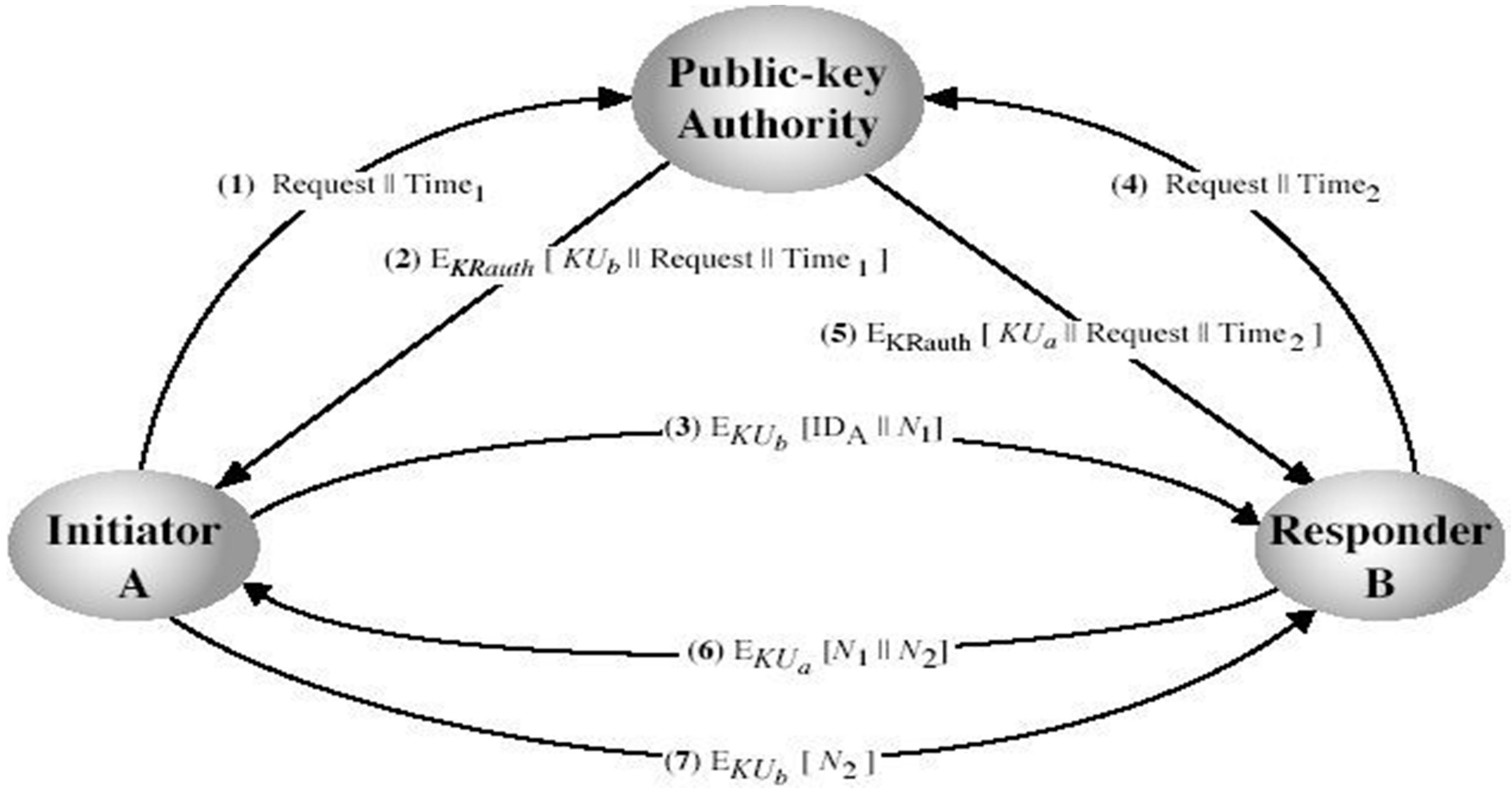
Key Management

- public-key encryption helps address key distribution problems
- distribution of public keys
- use of public-key encryption to distribute secret keys

Distribution of Public Keys

- can be considered as using one of:
  - Public announcement
  - Publicly available directory
  - Public-key authority
  - Public-key certificates

Public-Key Authority


Public-key Authority
Initiator A
Responder B
(1) Request ‖ Time$_1$
(2) E_{KRauth} [KU_b ‖ Request ‖ Time$_1$]
(3) E_{KU_b} [ID_A ‖ N_1]
(4) Request ‖ Time$_2$
(5) E_{KRauth} [KU_a ‖ Request ‖ Time$_2$]
(6) E_{KU_a} [N_1 ‖ N_2]
(7) E_{KU_b} [N_2]

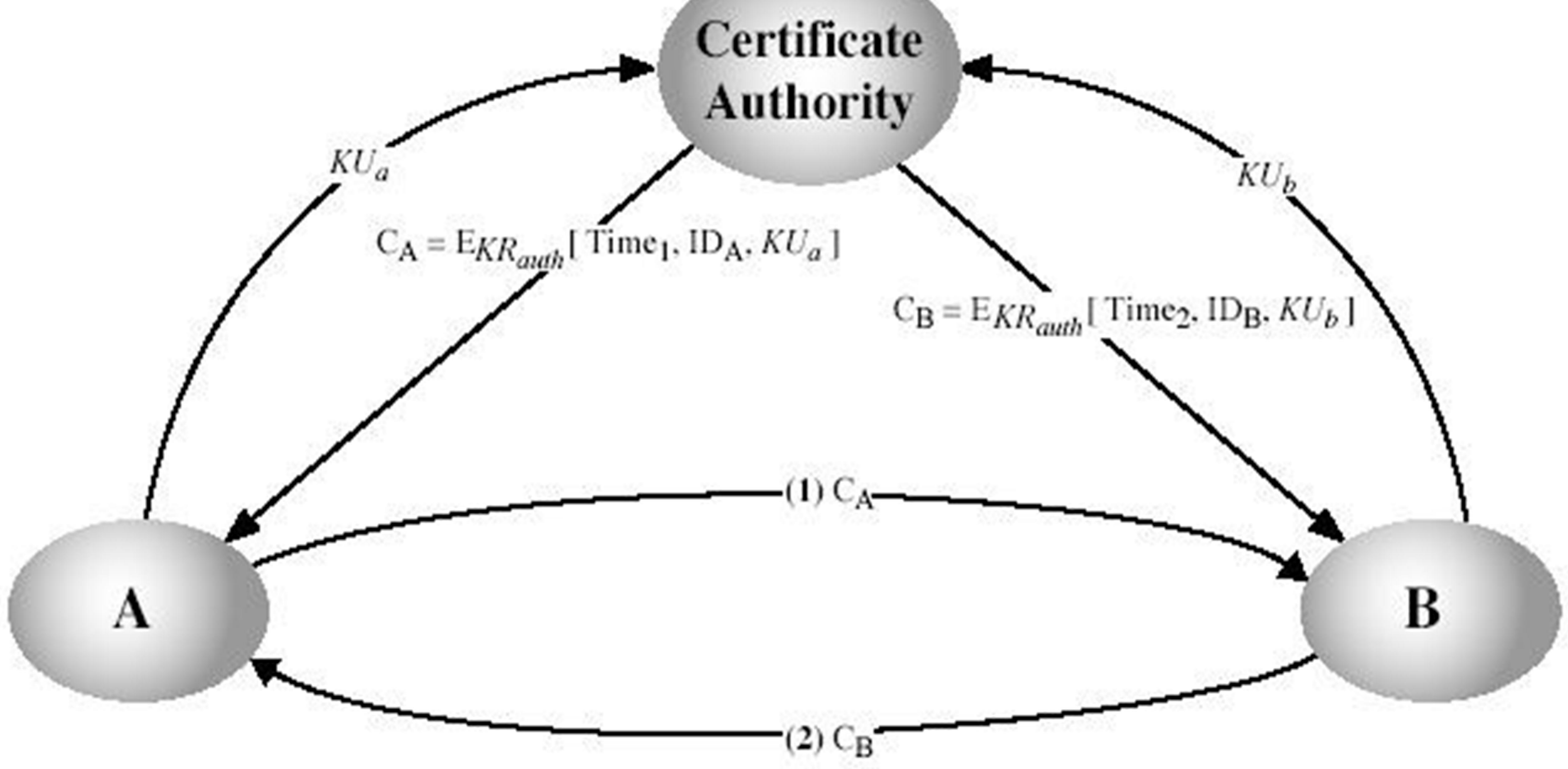
Public-Key Authority

- improve security by tightening control over distribution of keys from directory
- 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 Certificates

- The public-key authority could be a bottleneck in the system.
  - must appeal to the authority for the key of every other user
- certificates allow key exchange without real-time access to public-key authority
- a certificate binds **identity** to **public key**
- with all contents **signed** by a trusted Public-Key or Certificate Authority (CA)
  - Certifies the identity
  - Only the CA can make the certificates

Public-Key Certificates



Certificate Authority
KU_a
KU_b
$C_A = E_{KR_{auth}}[\text{Time}_1, ID_A, KU_a]$
$C_B = E_{KR_{auth}}[\text{Time}_2, ID_B, KU_b]$
(1) C_A
(2) C_B
A
B

Public-Key Distribution of Secret Keys

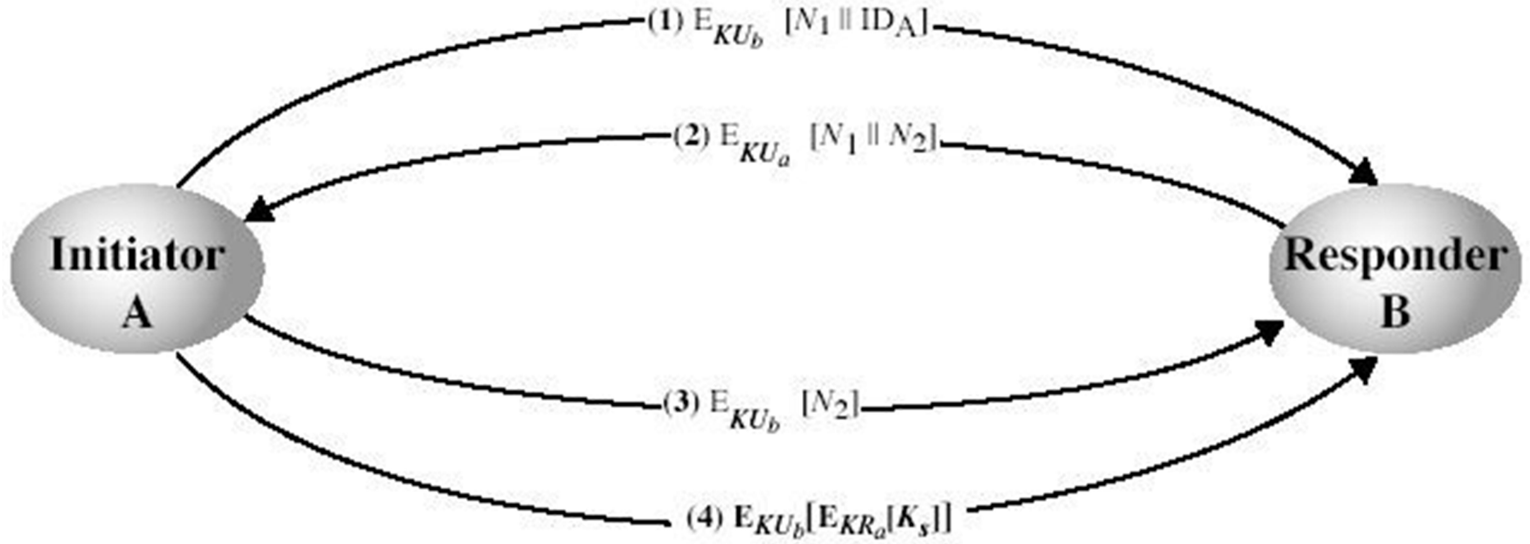
- public-key algorithms are slow
- so usually want to use private-key encryption to protect message contents
- hence need a session key
- have several alternatives for negotiating a suitable session using public-key

Simple Secret Key Distribution

- proposed by Merkle in 1979
  - A generates a new temporary public key pair
  - A sends B the public key and their identity
  - B generates a session key K sends it to A encrypted using the supplied public key
  - A decrypts the session key and both use
- problem is that an opponent can intercept and impersonate both halves of protocol

Public-Key Distribution of Secret Keys

- First securely exchanged public-keys using a previous method



Initiator A
Responder B
(1) E_{KU_b} $[N_1 \| ID_A]$
(2) E_{KU_a} $[N_1 \| N_2]$
(3) E_{KU_b} $[N_2]$
(4) $E_{KU_b}[E_{KR_a}[K_s]]$

Diffie-Hellman Key Exchange

- first public-key type scheme proposed
  - For key distribution only
- by Diffie & Hellman in 1976 along with the exposition of public key concepts
  - note: now know that James Ellis (UK CESG) secretly proposed the concept in 1970
- is a practical method for public exchange of a secret key

Diffie-Hellman Key Exchange

- a public-key distribution scheme
  - cannot be used to exchange an arbitrary message
  - rather it can establish a common key
  - known only to the two participants
- value of key depends on the participants (and their private and public key information)
- based on exponentiation in a finite (Galois) field (modulo a prime or a polynomial) - easy
- security relies on the difficulty of computing discrete logarithms (similar to factoring) – hard

Diffie-Hellman Setup

- all users agree on global parameters:
  - large prime integer or polynomial q
  - α a primitive root mod q
- each user (eg. A) generates their key
  - chooses a secret key (number): $x_A < q$
  - compute their **public key**: $y_A = \alpha^{xA} \bmod q$
- each user makes public that key y_A

Diffie-Hellman Key Exchange

- shared session key for users A & B is K:
  $K = y_A^{x_B} \bmod q$ (which **B** can compute)
  $K = y_B^{x_A} \bmod q$ (which **A** can compute)
  (example)
- K 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

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:

$K_{AB} = y_B^{xA} \bmod 353 = 248^{97} = 160$ (Alice)

$K_{AB} = y_A^{xB} \bmod 353 = 40^{233} = 160$ (Bob)

Elliptic Curve Cryptography

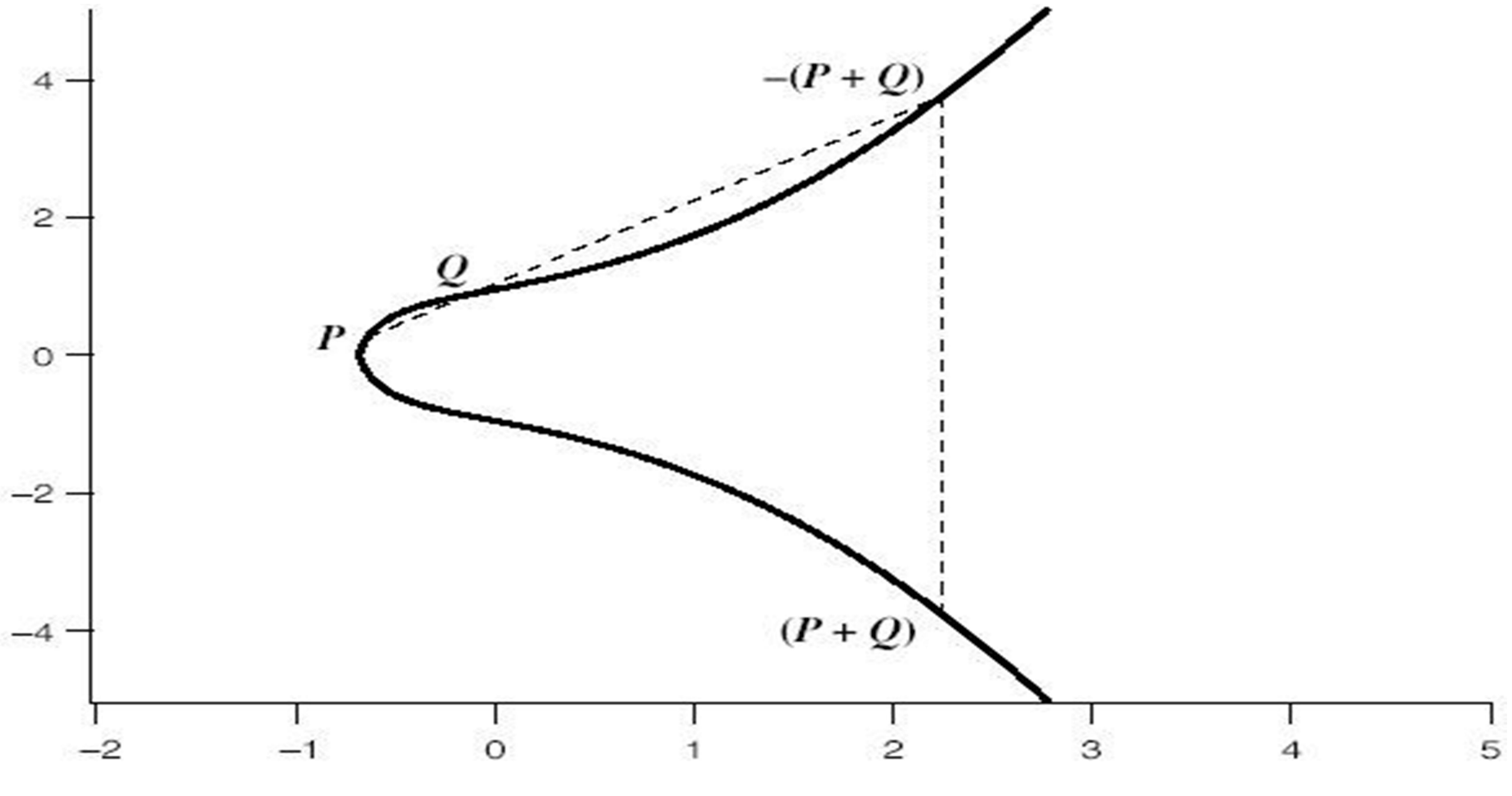
- majority of public-key crypto (RSA, D-H) use either integer or polynomial arithmetic with very large numbers/polynomials
- imposes a significant load in storing and processing keys and messages
- an alternative is to use elliptic curves
- offers same security with smaller bit sizes

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
- have addition operation for elliptic curve
  - Q+R is reflection of intersection R
  - Closed form for additions
    - (10.3) and (10.4) P.300-301

Real Elliptic Addition

Rule 1-5 in P.300



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

Finite Elliptic Curves

- Elliptic curve cryptography uses curves whose variables & coefficients are finite integers
- have two families commonly used:
  - prime curves $E_p(a,b)$ defined over Z_p
    - $y^2 \bmod p = (x^3+ax+b) \bmod p$
    - use integers modulo a prime for both variables and coeff
    - best in software
  - Closed form of additions: P.303
  - Example: P=(3,10), Q=(9,7), in $E_{23}(1,1)$
    - P+Q = (17,20)
    - 2P = (7,12)

Finite Elliptic Curves

- have two families commonly used:
  - binary curves $E_{2m}(a,b)$ defined over $GF(2^m)$
    - use polynomials with binary coefficients
    - best in hardware
  - Take a slightly different form of the equation
  - Different close forms for addition (P.304)

Elliptic Curve Cryptography

- ECC addition is analog of multiply
- ECC repeated addition is analog of exponentiation
- need “hard” problem equiv to discrete log
  - Q=kP, where Q,P are points in an elliptic 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: $E_{23}(9,17)$ (P.305)
  - k could be so large as to make brute-force fail

ECC Key Exchange

- can do key exchange similar to D-H
- users select a suitable curve $E_p(a,b)$
  - Either a prime curve, or a binary curve
- 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: $P_A=n_A\times G$, $P_B=n_B\times G$
- compute shared key: $K=n_A\times P_B$, $K=n_B\times P_A$
  - same since $K=n_A\times n_B\times G$
- Example: P.305

ECC Encryption/Decryption

- select suitable curve & point G as in D-H
- encode any message M as a point on the elliptic curve $P_m=(x,y)$
- each user chooses private key $n_A<n$
- and computes public key $P_A=n_A\times G$
- to encrypt pick random k: $C_m=\{kG,\ P_m+k\ P_b\}$,
- decrypt C_m compute:

  $P_m+kP_b-n_B(kG) = P_m+k(n_BG)-n_B(kG) = P_m$
- Example: P.307

Table 10.2 Computational Effort for Cryptanalysis of Elliptic Curve Cryptography Compared to RSA

Key Size	MIPS-Years
150	3.8×10^{10}
205	7.1×10^{18}
234	1.6×10^{28}

(a) Elliptic Curve Logarithms using the Pollard rho Method

Key Size	MIPS-Years
512	3×10^{4}
768	2×10^{8}
1024	3×10^{11}
1280	1×10^{14}
1536	3×10^{16}
2048	3×10^{20}

(b) Integer Factorization using the General Number Field Sieve

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

Message Authentication

- protecting message content (ie secrecy) by encrypting the message
- now consider
  - how to protect message integrity (ie protection from modification)
  - confirming the identity of the sender
- then three alternative functions used:
  - message encryption (the ciphertext itself is the authenticator)
  - message authentication code (MAC)
  - hash function

Security Attacks

- disclosure of message contents
- traffic analysis (discover the pattern)
- Masquerade (insert a msg from a fraudulent source)
- content modification
- sequence modification (insert, delete, reorder)
- timing modification (delay or replay)
- source repudiation (denial of a transmission)
- destination repudiation (denial of a receipt)

Message Encryption

- message encryption by itself also provides a measure of authentication
- if symmetric encryption is used then:
  - receiver know sender must have created it
  - since only sender and receiver now key used
  - know content cannot of been altered
  - if message has suitable structure, redundancy or a checksum to detect any changes

Message Encryption

- if public-key encryption is used:
  - encryption provides no confidence of sender
  - since anyone potentially knows public-key
  - however if
    - sender **signs** message using their private-key
    - then encrypts with recipients public key
    - have both secrecy and authentication
  - again need to recognize corrupted messages
  - but at cost of two public-key uses on message

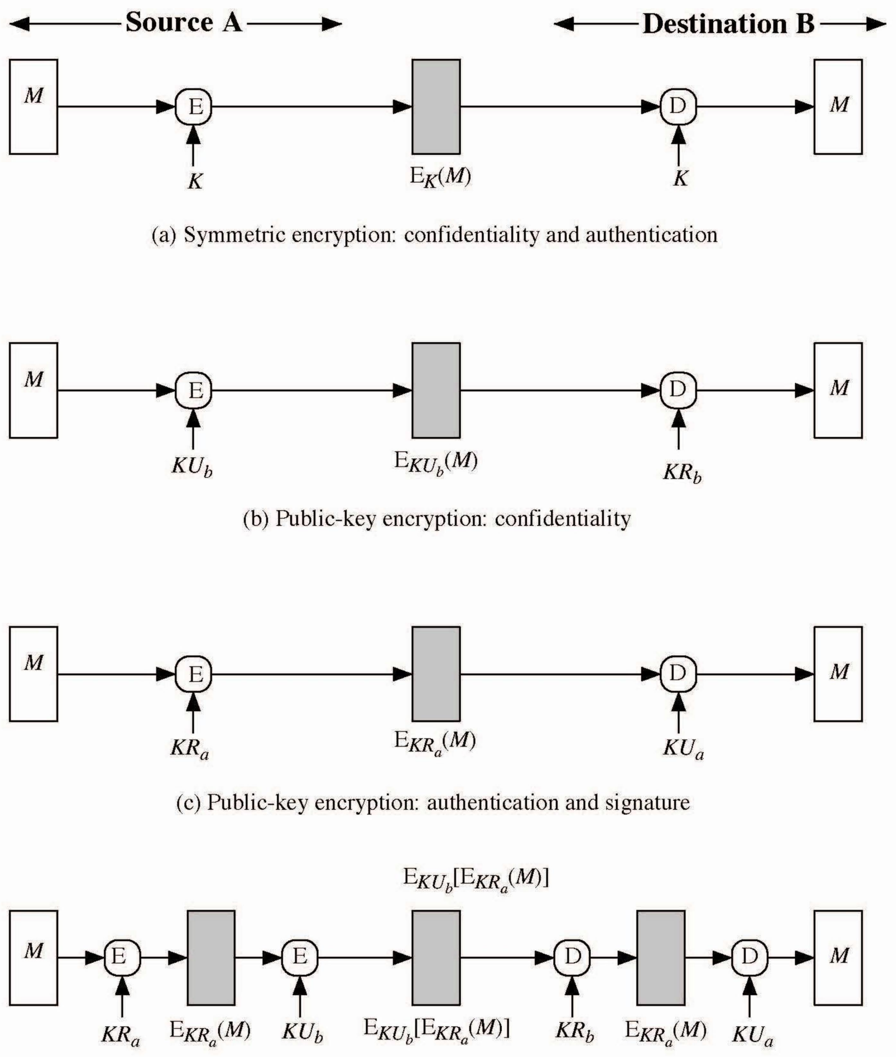
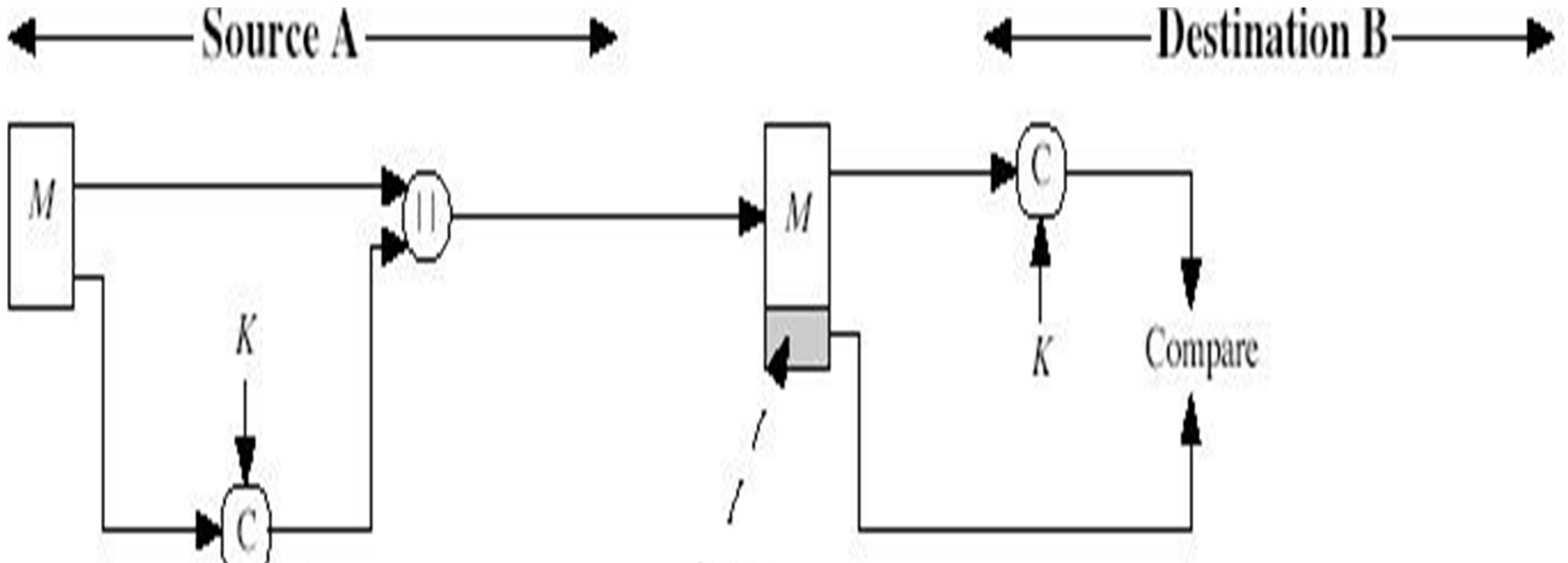

Source A
Destination B
M
E
K
$E_K(M)$
D
K
M
(a) Symmetric encryption: confidentiality and authentication
M
E
KU_b
$E_{KU_b}(M)$
D
KR_b
M
(b) Public-key encryption: confidentiality
M
E
KR_a
$E_{KR_a}(M)$
D
KU_a
M
(c) Public-key encryption: authentication and signature
$E_{KU_b}[E_{KR_a}(M)]$
M
E
KR_a
$E_{KR_a}(M)$
E
KU_b
$E_{KU_b}[E_{KR_a}(M)]$
D
KR_b
$E_{KR_a}(M)$
D
KU_a
M
(d) Public-key encryption: confidentiality, authentication, and signature

Figure 11.1 Basic Uses of Message Encryption

Message Authentication Code (MAC)

- generated by an MAC function C that creates a small fixed-sized block
  - depending on both message M and a shared secret key K, $MAC=C_K(M)$
  - MAC is appended to the message M
- receiver performs same computation on message and checks it matches the MAC
- provides assurance that message is unaltered and comes from sender

Message Authentication Code



(a) Message authentication

Message Authentication Codes

- can also use encryption for secrecy
  - generally use separate keys for each
  - can compute MAC either before or after encryption
  - is generally regarded as better done before
- why use a MAC?
  - MAC is much less expensive than en/decryption
  - sometimes only authentication is needed
  - One end with a heavy load, check MAC selectively

MAC Properties

- a MAC is a cryptographic checksum

  MAC = $C_K(M)$
  - condenses a variable-length message M
  - using a secret key K
  - to a fixed-sized authenticator
- is a many-to-one function
  - potentially many messages have same MAC
    - 100-bit M, and 20-bit MAC

Requirements for MACs

- taking into account the types of attacks
- need the MAC to satisfy the following:
  1. knowing a message and MAC, is infeasible to find another message with same MAC
  2. MACs should be uniformly distributed
  3. MAC should depend equally on all bits of the message

Using Symmetric Ciphers for MACs

- can use any block cipher chaining mode and use final block as a MAC
- **Data Authentication Algorithm (DAA)** is a widely used MAC based on DES-CBC
  - using IV=0 and zero-pad of final block
  - encrypt message using DES in CBC mode
  - and send just the final block as the MAC
    - or the leftmost M bits ($16 \le M \le 64$) of final block

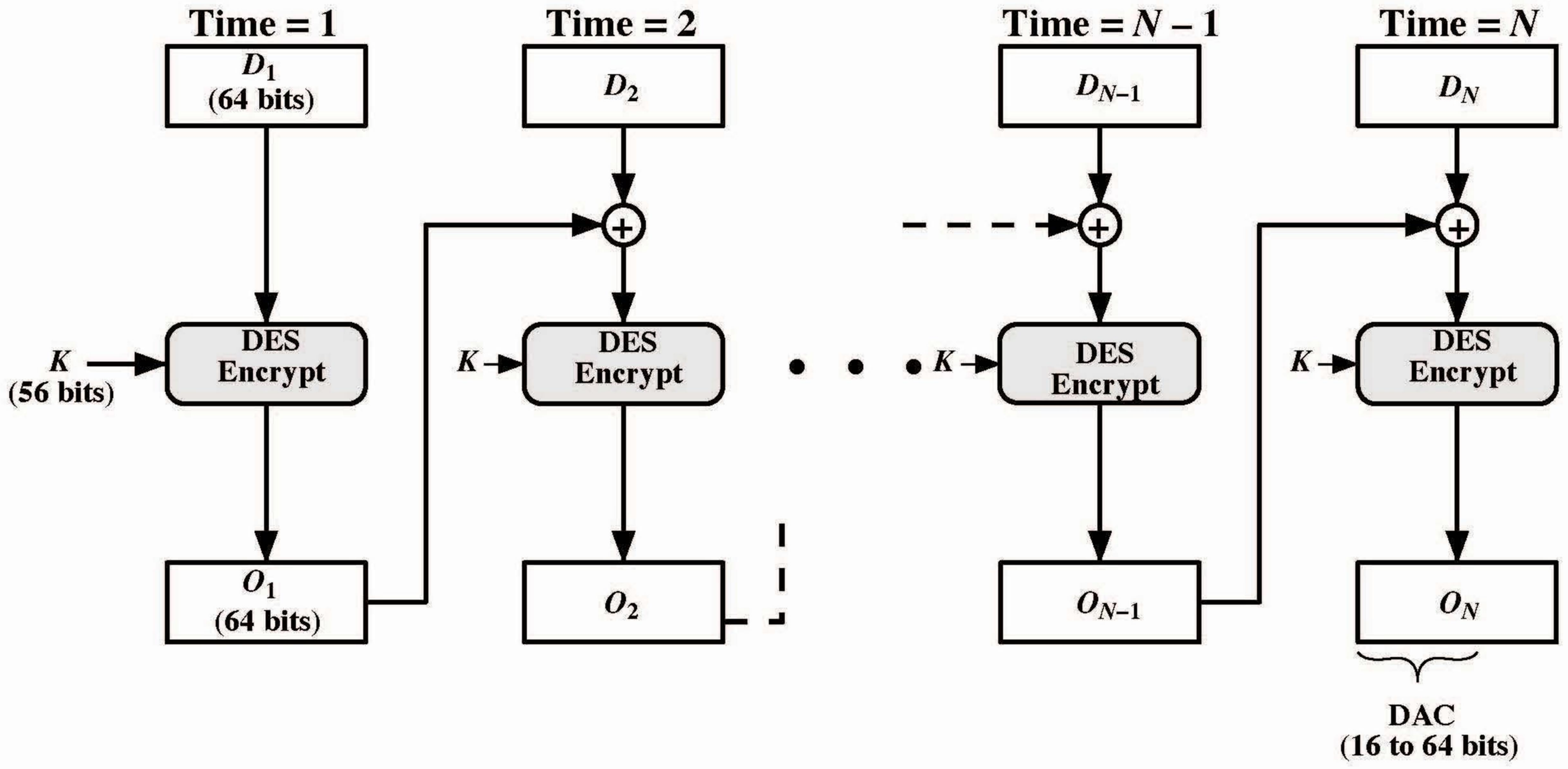
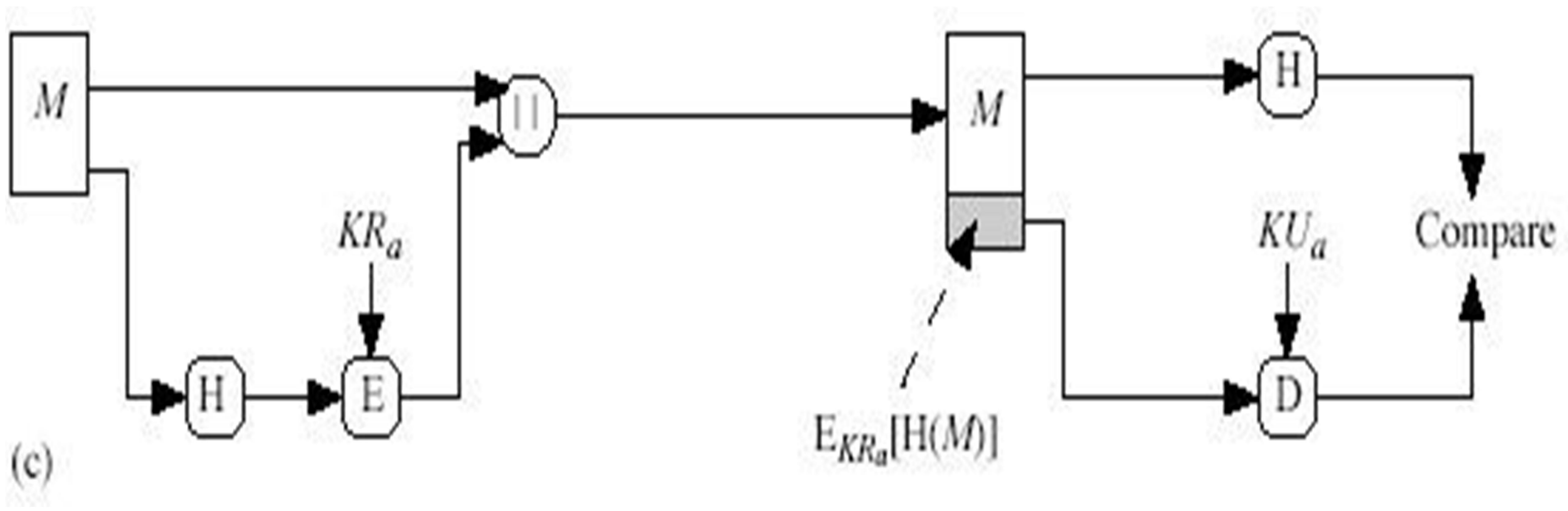


Figure 11.6 Data Authentication Algorithm (FIPS PUB 113)

Hash Functions

- condenses arbitrary message to fixed size
- usually assume that the hash function is public and not keyed
  - cf. MAC which is keyed
- used to detect changes to message
- can use in various ways with message
- most often to create a digital signature

Hash Functions & Digital Signatures



Hash Function Properties

- a Hash Function produces a fingerprint of some file/message/data

  $h = H(M)$

  – condenses a variable-length message M

  – to a fixed-sized fingerprint
- assumed to be public

Requirements for Hash Functions

1. can be applied to any sized message M
2. produces fixed-length output h
3. is easy to compute h=H(M) for any message M
4. given h is infeasible to find x s.t. H(x)=h
   - one-way property
5. given x is infeasible to find y s.t. H(y)=H(x)
   - weak collision resistance
6. is infeasible to find any x,y s.t. H(y)=H(x)
   - strong collision resistance

Block Ciphers as Hash Functions

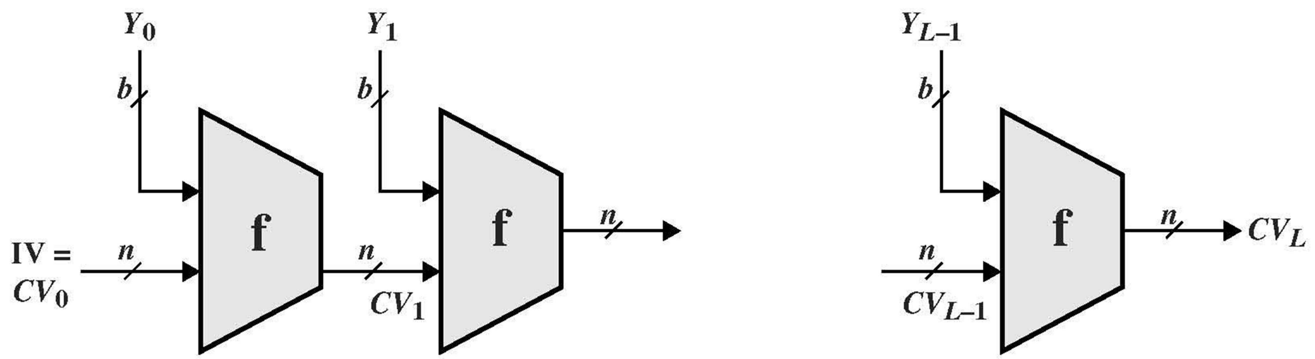
- can use block ciphers as hash functions
  - using H_0=0 and zero-pad of final block
  - compute: $H_i = E_{M_i} [H_{i-1}]$
  - and use final block as the hash value
  - similar to CBC but without a key
- resulting hash is too small (64-bit)
  - due to direct birthday attack and variants

Hash Functions & MAC Security

- like block ciphers have:
- **brute-force** attacks exploiting
  - strong collision resistance hash have cost $2^{m/2}$
    - 128-bit hash looks vulnerable, 160-bits better
  - MACs with known message-MAC pairs
    - can either attack keyspace (cf key search) or MAC
      - Min(2^k, 2^n)
    - at least 128-bit MAC and 128-bit key is needed for security

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



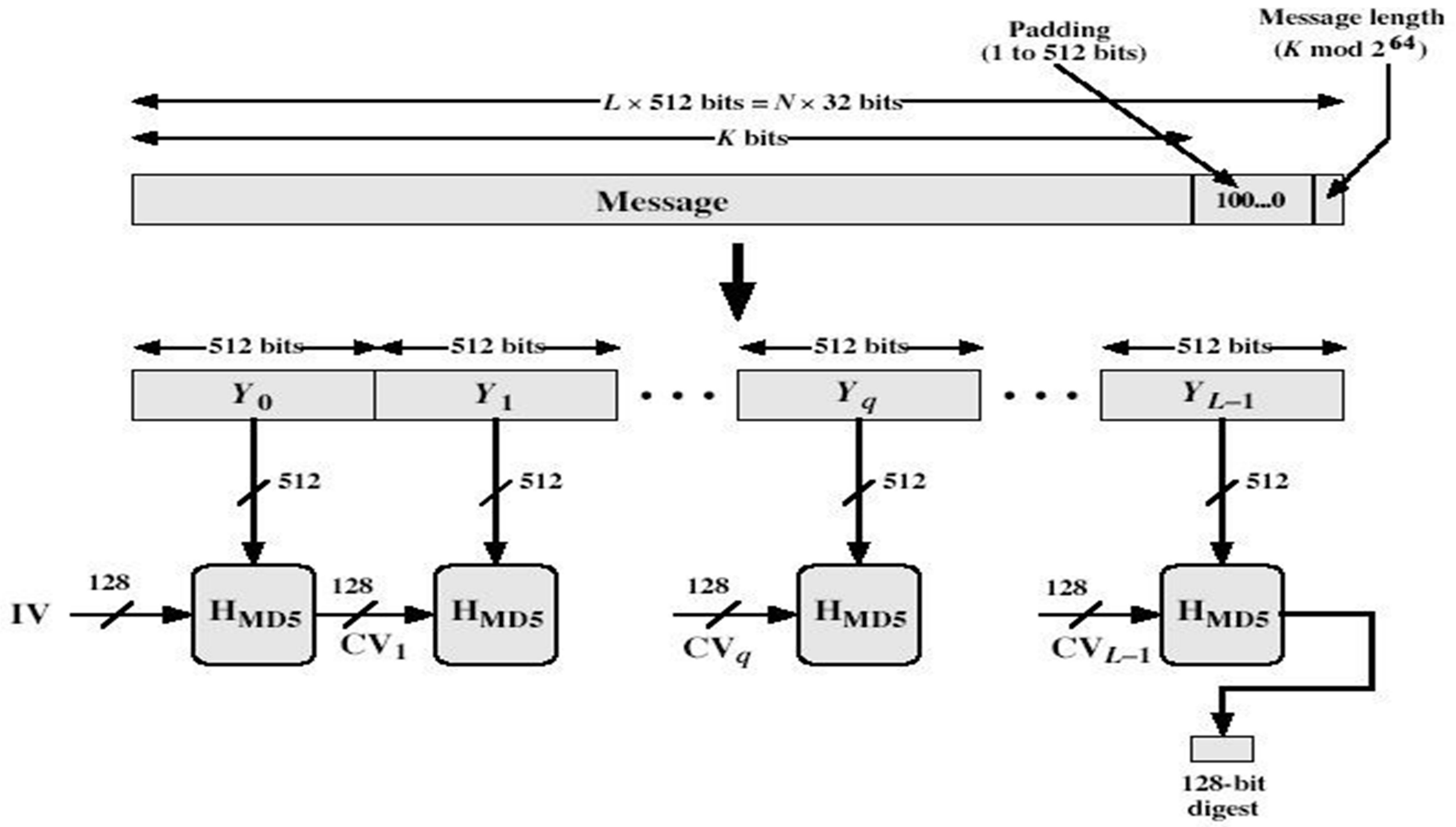
IV	=	Initial value
CV	=	chaining variable
Y_i	=	ith input block
f	=	compression algorithm
L	=	number of input blocks
n	=	length of hash code
b	=	length of input block

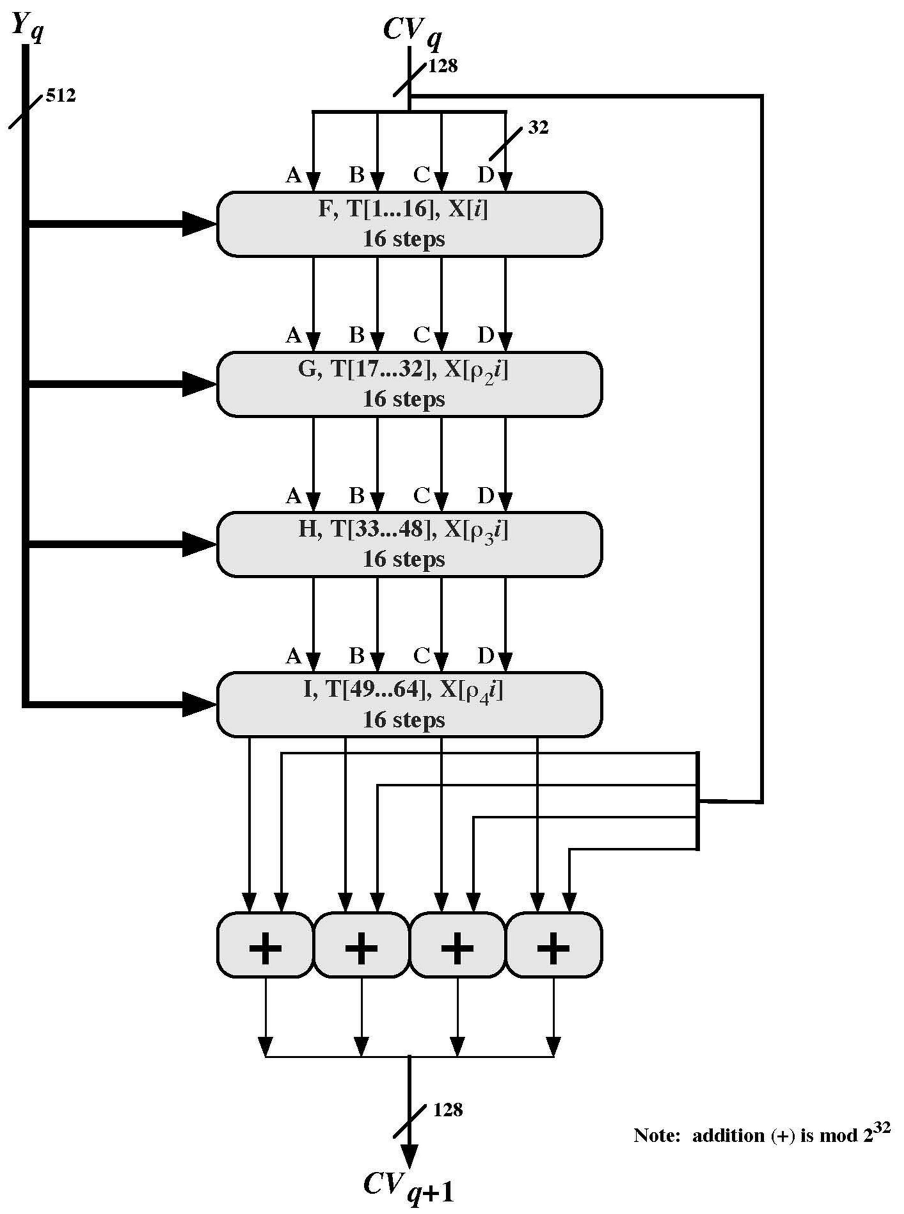
Figure 11.10 General Structure of Secure Hash Code

MD5

- 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





Y_q
CV_q
128
512
32
A
B
C
D
F, T[1...16], X[i]
16 steps
G, T[17...32], X[$\rho_2 i$]
16 steps
H, T[33...48], X[$\rho_3 i$]
16 steps
I, T[49...64], X[$\rho_4 i$]
16 steps
+
128
Note: addition (+) is mod 2^{32}
CV_{q+1}

Figure 12.2 MD5 Processing of a Single 512-bit Block

MD5 Compression Function

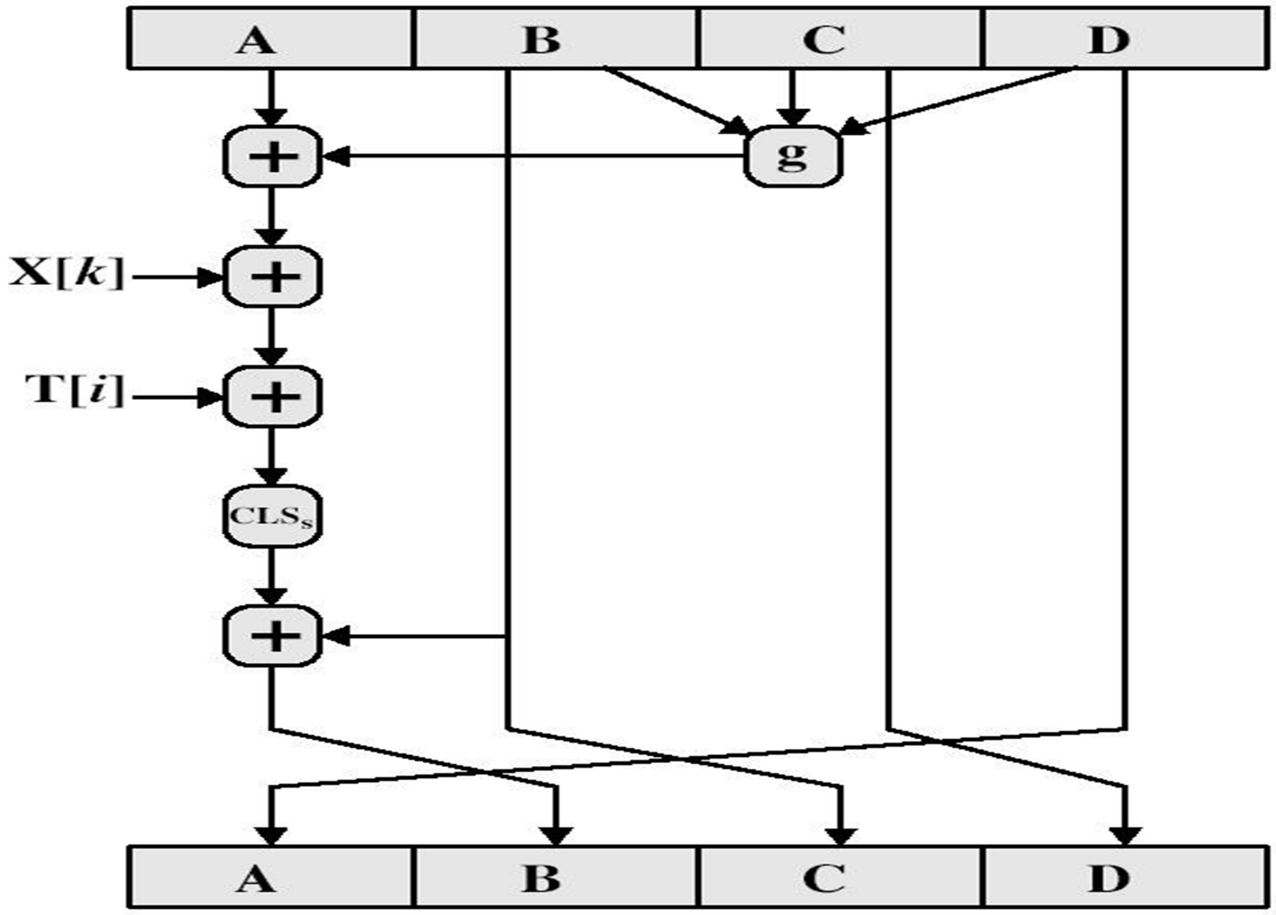


Table 12.1 Key Elements of MD5

(a) Truth table of logical functions

b	c	d	F	G	H	I
0	0	0	0	0	0	1
0	0	1	1	0	1	0
0	1	0	0	1	1	0
0	1	1	1	0	0	1
1	0	0	0	0	1	1
1	0	1	0	1	0	1
1	1	0	1	1	0	0
1	1	1	1	1	1	0

(b) Table T, constructed from the sine function

T[1] = D76AA478	T[17] = F61E2562	T[33] = FFFA3942	T[49] = F4292244
T[2] = E8C7B756	T[18] = C040B340	T[34] = 8771F681	T[50] = 432AFF97
T[3] = 242070DB	T[19] = 265E5A51	T[35] = 699D6122	T[51] = AB9423A7
T[4] = C1BDCEEE	T[20] = E9B6C7AA	T[36] = FDE5380C	T[52] = FC93A039
T[5] = F57C0FAF	T[21] = D62F105D	T[37] = A4BEEA44	T[53] = 655B59C3
T[6] = 4787C62A	T[22] = 02441453	T[38] = 4BDECFA9	T[54] = 8F0CCC92
T[7] = A8304613	T[23] = D8A1E681	T[39] = F6BB4B60	T[55] = FFEFF47D
T[8] = FD469501	T[24] = E7D3FBC8	T[40] = BEBFBC70	T[56] = 85845DD1
T[9] = 698098D8	T[25] = 21E1CDE6	T[41] = 289B7EC6	T[57] = 6FA87E4F
T[10] = 8B44F7AF	T[26] = C33707D6	T[42] = EAA127FA	T[58] = FE2CE6E0
T[11] = FFFF5BB1	T[27] = F4D50D87	T[43] = D4EF3085	T[59] = A3014314
T[12] = 895CD7BE	T[28] = 455A14ED	T[44] = 04881D05	T[60] = 4E0811A1
T[13] = 6B901122	T[29] = A9E3E905	T[45] = D9D4D039	T[61] = F7537E82
T[14] = FD987193	T[30] = FCEFA3F8	T[46] = E6DB99E5	T[62] = BD3AF235
T[15] = A679438E	T[31] = 676F02D9	T[47] = 1FA27CF8	T[63] = 2AD7D2BB
T[16] = 49B40821	T[32] = 8D2A4C8A	T[48] = C4AC5665	T[64] = EB86D391

```
For q = 0 to (N/16)-1 do
    /* Copy block q into X. */
    For j = 0 to 15 do
        Set X[j] to M[q*16 + j].
    end /* of loop on j */

    /* Save A as AA, B as BB, C as CC, and
    D as DD. */
    AA = A
    BB = B
    CC = C
    DD = D

    /* Round 1. */
    /* Let [abcd k s i] denote the operation
    a = b + ((a + F(b,c,d) + X[k] + T[i]) <<<s).
    Do the following 16 operations. */
    [ABCD    0    7    1]
    [DABC    1   12    2]
    [CDAB    2   17    3]
    [BCDA    3   22    4]
    [ABCD    4    7    5]
    [DABC    5   12    6]
    [CDAB    6   17    7]
    [BCDA    7   22    8]
    [ABCD    8    7    9]
    [DABC    9   12   10]
    [CDAB   10   17   11]
    [BCDA   11   22   12]
    [ABCD   12    7   13]
    [DABC   13   12   14]
    [CDAB   14   17   15]
    [BCDA   15   22   16]

    /* Round 2. */
    /* Let [abcd k s i] denote the operation
    a = b + ((a + G(b,c,d) + X[k] + T[i]) <<<s).
    Do the following 16 operations. */
    [ABCD    1    5   17]
    [DABC    6    9   18]
    [CDAB   11   14   19]
    [BCDA    0   20   20]
    [ABCD    5    5   21]
    [DABC   10    9   22]
    [CDAB   15   14   23]
    [BCDA    4   20   24]
    [ABCD    9    5   25]
    [DABC   14    9   26]
    [CDAB    3   14   27]
    [BCDA    8   20   28]
    [ABCD   13    5   29]
    [DABC    2    9   30]
    [CDAB    7   14   31]
    [BCDA   12   20   32]

    /* Let [abcd k s i] denote the operation
    a = b + ((a + H(b,c,d) + X[k] + T[i]) <<<s).
    Do the following 16 operations. */
    [ABCD    5    4   33]
    [DABC    8   11   34]
    [CDAB   11   16   35]
    [BCDA   14   23   36]
    [ABCD    1    4   37]
    [DABC    4   11   38]
    [CDAB    7   16   39]
    [BCDA   10   23   40]
    [ABCD   13    4   41]
    [DABC    0   11   42]
    [CDAB    3   16   43]
    [BCDA    6   23   44]
    [ABCD    9    4   45]
    [DABC   12   11   46]
    [CDAB   15   16   47]
    [BCDA    2   23   48]

    /* Round 4. */
    /* Let [abcd k s i] denote the operation
    a = b + ((a + I(b,c,d) + X[k] + T[i]) <<<s).
    Do the following 16 operations. */
    [ABCD    0    6   49]
    [DABC    7   10   50]
    [CDAB   14   15   51]
    [BCDA    5   21   52]
    [ABCD   12    6   53]
    [DABC    3   10   54]
    [CDAB   10   15   55]
    [BCDA    1   21   56]
    [ABCD    8    6   57]
    [DABC   15   10   58]
    [CDAB    6   15   59]
    [BCDA   13   21   60]
    [ABCD    4    6   61]
    [DABC   11   10   62]
    [CDAB    2   15   63]
    [BCDA    9   21   64]

    /* Then increment each of the four registers by the
    value it had before this block was started. */
    A = A + AA
    B = B + BB
    C = C + CC
    D = D + DD

end /* of loop on q */
```

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 refer to the 4 words of the buffer, but used in varying permutations
  - note this updates 1 word only of the buffer
  - after 16 steps each word is updated 4 times
- where g(b,c,d) is a different nonlinear function in each round (F,G,H,I)
- T[i] is a constant value derived from sin
- The point of all this complexity:

Strength of MD5

- Every hash bit is dependent on all message bits
- Rivest conjectures security is as good as possible for a 128 bit hash
  - Given a hash, find a message: O($2^{128)}$ operations
  - No disproof exists yet
- known attacks are:
  - Berson 92 attacked any 1 round using differential cryptanalysis (but can't extend)
  - Boer & Bosselaers 93 found a pseudo collision (again unable to extend)
  - Dobbertin 96 created collisions on MD compression function for one block, cannot expand to many blocks
  - Brute-force search now considered possible

Secure Hash Algorithm (SHA-1)

- 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 (67452301,efcdab89,98badcfe,10325476,c3d2e1f0)
4. process message in 16-word (512-bit) chunks:
   – expand 16 words into 80 words by mixing & shifting
   – use 4 rounds of 20 bit operations on message block & buffer
   – add output to input to form new buffer value
5. output hash value is the final buffer value

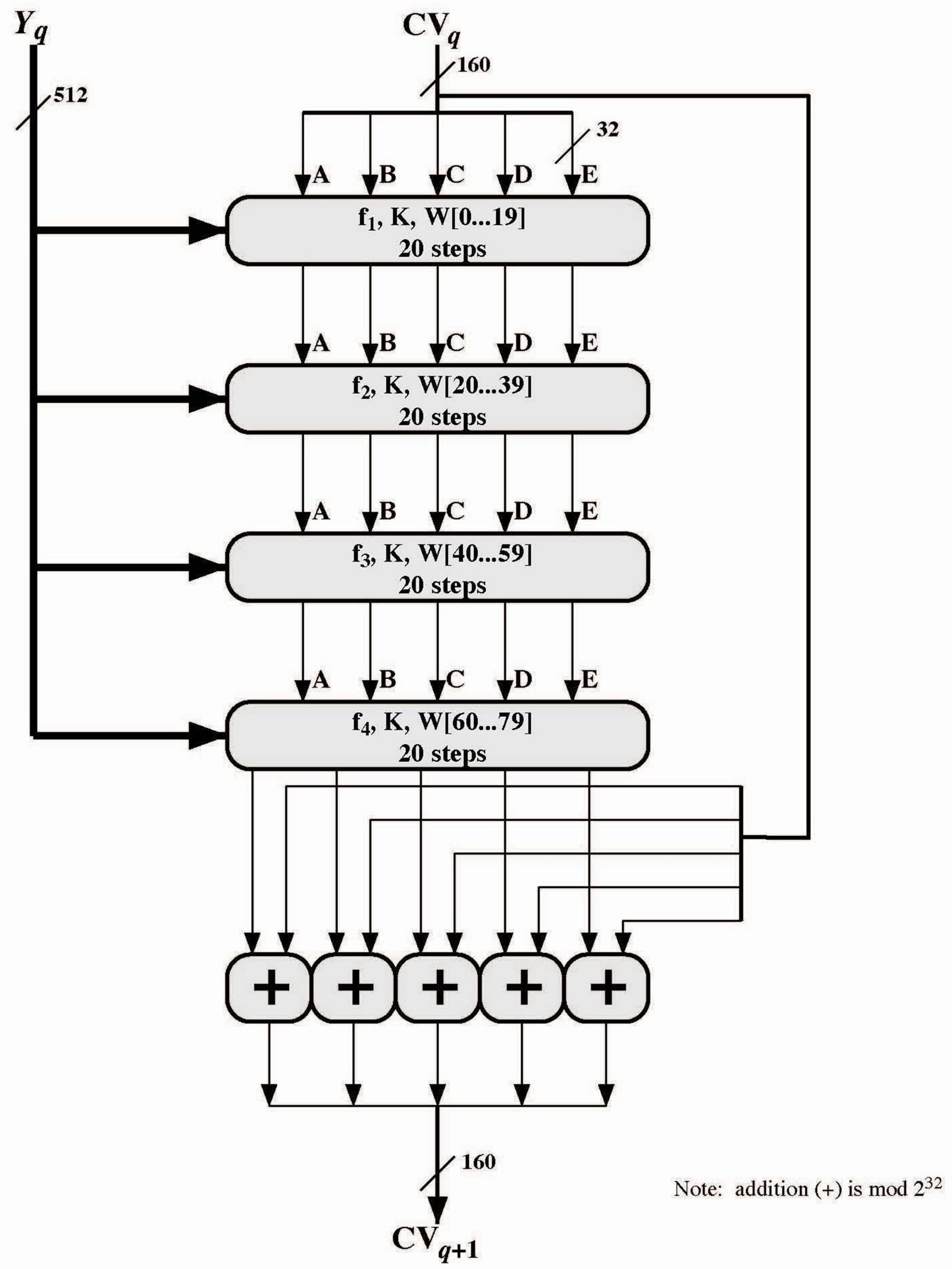
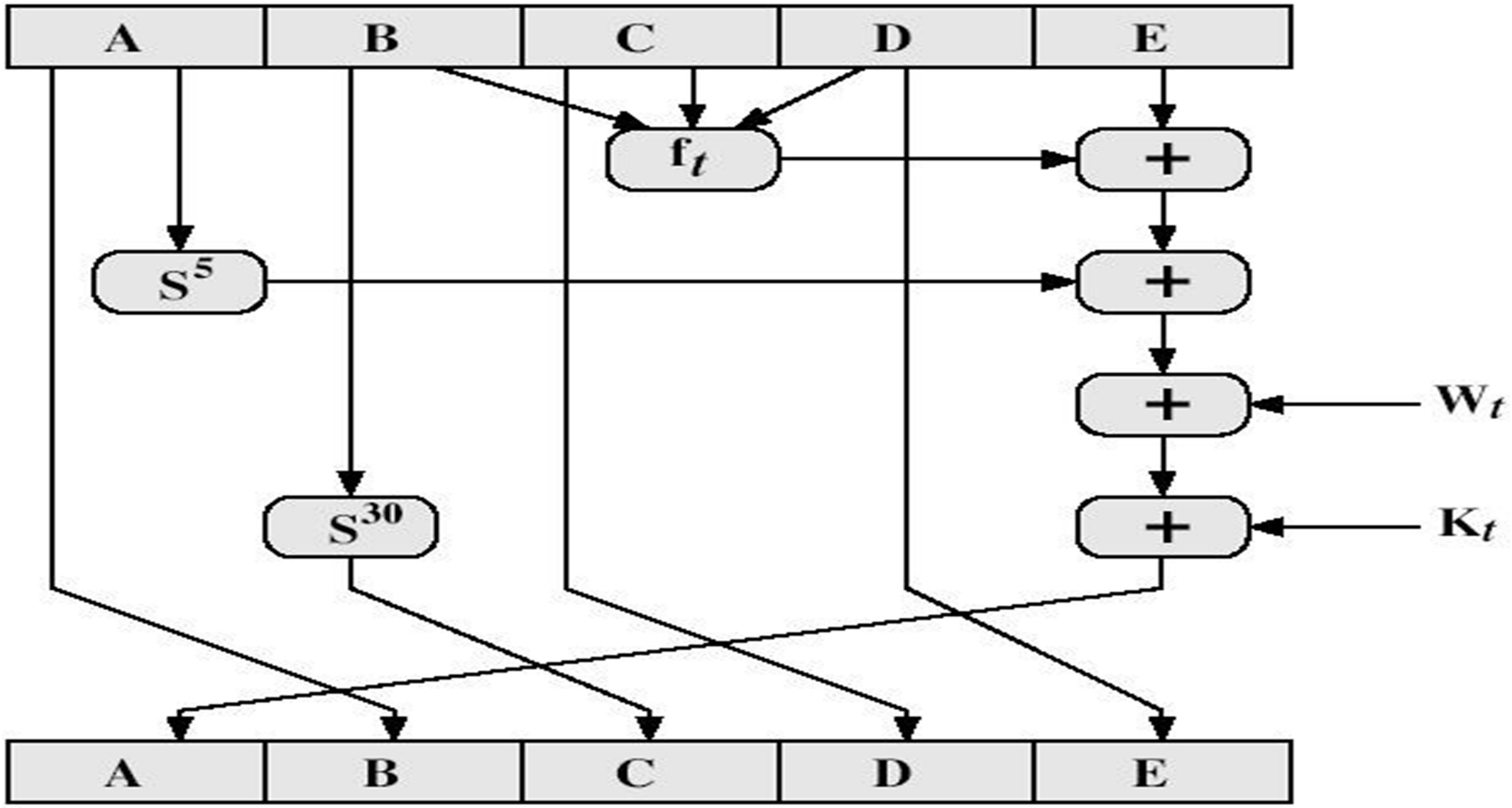


Figure 12.5 SHA-1 Processing of a Single 512-bit Block (SHA-1 Compression Function)

SHA-1 Compression Function



Logical functions for SHA-1

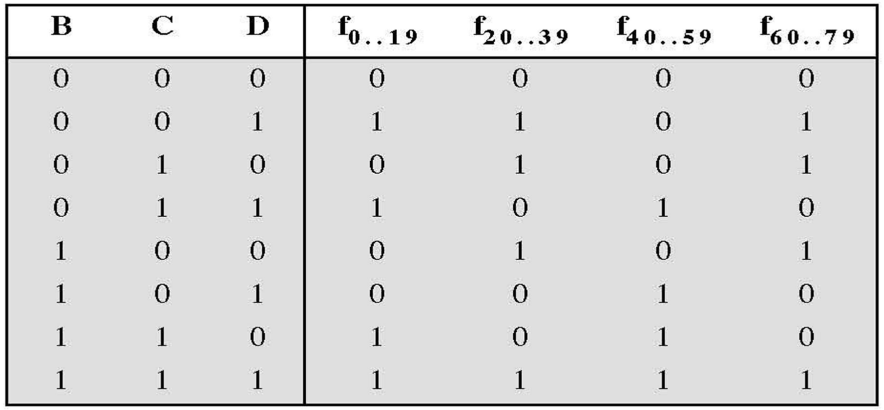


Table 12.2 Truth Table of Logical Functions for SHA-1

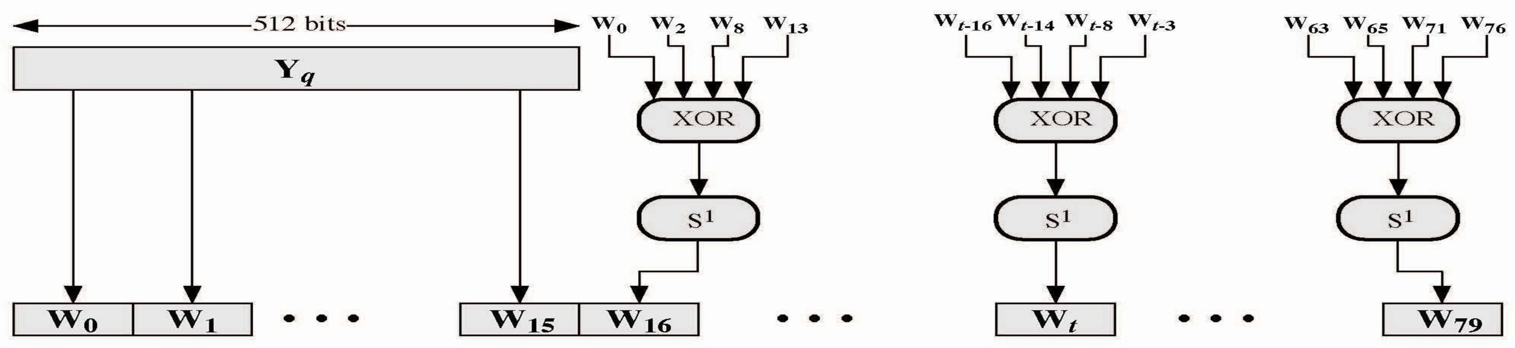
B	C	D	$f_{0..19}$	$f_{20..39}$	$f_{40..59}$	$f_{60..79}$
0	0	0	0	0	0	0
0	0	1	1	1	0	1
0	1	0	0	1	0	1
0	1	1	1	0	1	0
1	0	0	0	1	0	1
1	0	1	0	0	1	0
1	1	0	1	0	1	0
1	1	1	1	1	1	1

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)
- ABCDE refer to the 5 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 (P359)

Creation of 80-word input

- Adds redundancy and interdependence among message blocks



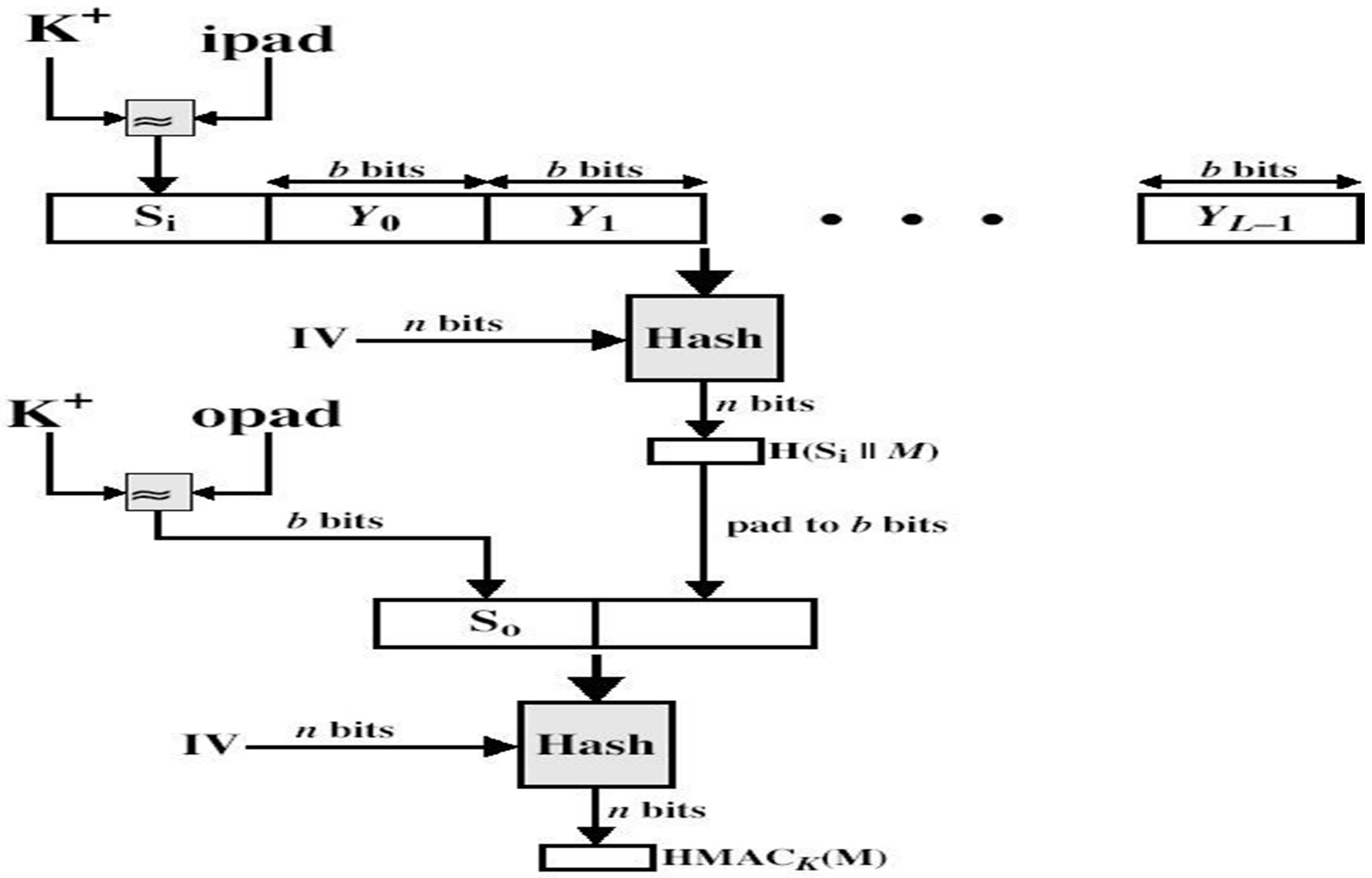
512 bits
Y_q
W_0 W_2 W_8 W_{13}
W_{t-16} W_{t-14} W_{t-8} W_{t-3}
W_{63} W_{65} W_{71} W_{76}
XOR
S^1
W_0 W_1 • • • W_{15} W_{16} • • • W_t • • • W_{79}

Figure 12.7 Creation of 80-word Input Sequence for SHA-1 Processing of Single Block

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 (SUN) vs MD5 for little endian CPU’s (PC)

HMAC Overview



HMAC

- specified as Internet standard RFC2104
- uses hash function on the message:

  $HMAC_K = Hash[(K^+ \text{ XoR } opad) \,||\, Hash[(K^+ \text{ XoR } ipad)||M)]]$
- where K^+ is the key padded out to size
- and opad, ipad are specified padding constants
- overhead is just 3 more hash calculations than the message needs alone
- any of MD5, SHA-1, RIPEMD-160 can be used

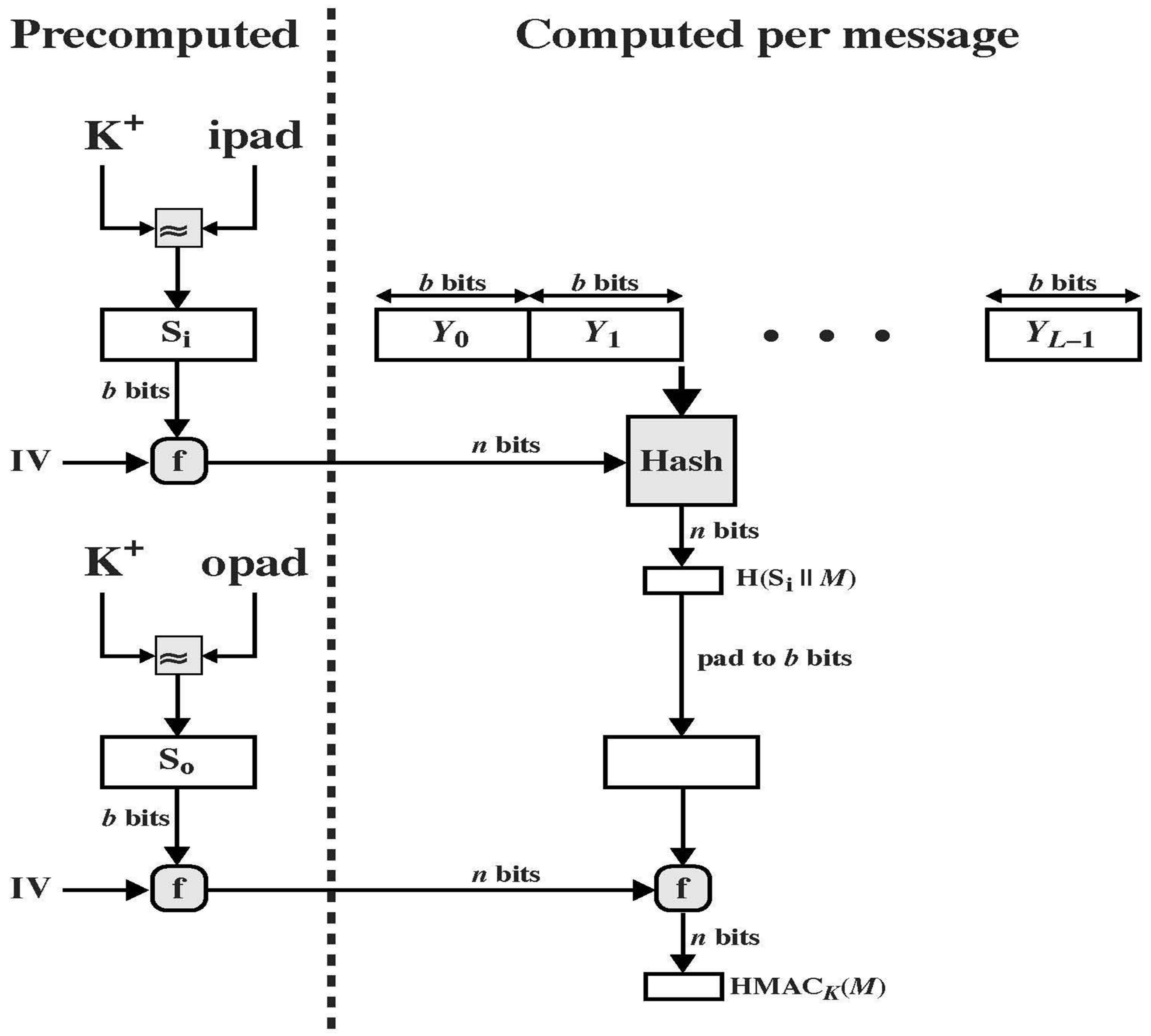


Figure 12.11 Efficient Implementation of HMAC

Digital Signatures

- have looked at message authentication
  - but does not address issues of lack of trust
  - Mary may forge a message and claim it came from John
  - John can deny sending a meesage
- digital signatures provide the ability to:
  - verify author, date & time of signature
  - authenticate message contents
  - be verified by third parties to resolve disputes
- hence include authentication function with additional capabilities

Digital Signature Properties

- must depend on the message being signed
- must use information unique to sender
  - to prevent both forgery and denial
- must be relatively easy to produce
- must be relatively easy to recognize & verify
- be computationally infeasible to forge
  - with new message for existing digital signature
  - with fraudulent digital signature for given message
- be practical save a copy of the digital signature in storage

Direct Digital Signatures

- involve only sender & receiver
- assumed receiver has sender’s public-key
- digital signature made by sender signing entire message or hash with private-key
- can further encrypt using receivers public-key
- important that sign first then encrypt message & signature
- security depends on sender’s private-key
  - Have problems if lost/stolen

Arbitrated Digital Signatures

- involves use of arbiter A
  - validates any signed message
  - then dated and sent to recipient
- requires a great deal of trust in arbiter
- can be implemented with either private or public-key algorithms
- arbiter may or may not see message

Table 13.1 Arbitrated Digital Signature Techniques

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

Notation:

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

Authentication Protocols

- used to convince parties of each others identity and to exchange session keys
- may be one-way or mutual
- key issues are
  - confidentiality - to protect session keys
  - timeliness - to prevent replay attacks

Replay Attacks

- where a valid signed message is copied and later resent
  - simple replay
  - repetition that can be logged
  - repetition that cannot be detected
  - backward replay without modification
- countermeasures include
  - use of sequence numbers (generally impractical)
  - timestamps (needs synchronized clocks)
  - challenge/response (using unique nonce)

Using Symmetric Encryption

- as discussed previously can use a two-level hierarchy of keys
- usually with a trusted Key Distribution Center (KDC)
  - each party shares own master key with KDC
  - KDC generates session keys used for connections between parties
  - master keys used to distribute these to them

Needham-Schroeder Protocol

- original third-party key distribution protocol
- for session between A B mediated by KDC
- protocol overview is: Fig 7.9

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

2. KDC→A: $E_{Ka}[Ks \,||\, ID_B \,||\, N_1 \,||\, E_{Kb}[Ks||ID_A]\,]$

3. A→B: $E_{Kb}[Ks||ID_A]$

4. B→A: $E_{Ks}[N_2]$

5. A$\rightarrow$B: $E_{Ks}[\mathrm{f}(N_2)]$

One-Way Authentication

- required when sender & receiver are not in communications at same time (eg. email)
- have header in clear so can be delivered by email system
- may want contents of body protected & sender authenticated
  - The receiver wants some assurance of the identity of the alleged sender

Using Symmetric Encryption

- can refine use of KDC but can't have final exchange of nonces:
  1. A→KDC: $ID_A \,||\, ID_B \,||\, N_1$
  2. KDC→A: $E_{Ka}[Ks \,||\, ID_B \,||\, N_1 \,||\, E_{Kb}[Ks||ID_A]\,]$
  3. A→B: $E_{Kb}[Ks||ID_A] \,||\, E_{Ks}[M]$
- Only the intended recipient can read it
- Certain level of authentication of A
- does not protect against replays
  - could rely on timestamp in message, though email delays make this problematic

Public-Key Approaches

- have seen some public-key approaches
- if confidentiality is major concern, can use:

  $A \rightarrow B$: $E_{KUb}[Ks] \,||\, E_{Ks}[M]$
  - has encrypted session key, encrypted message
  - More efficient than simply $E_{KUb}[M]$
- if authentication is the primary concern

  use a digital signature with a digital certificate:
  - $A \rightarrow B$: $M \,||\, E_{KRa}[H(M)]$, problematic
  - Encrypt everything using receiver's public key
  - $A \rightarrow B$: $M \,||\, E_{KRa}[H(M)] \,||\, E_{KRas}[T||ID_A||KU_a]$
  - with message, signature, certificate

Kerberos

- trusted key server system from MIT
- provides centralised private-key third-party authentication in a distributed network
  - allows users access to services distributed through network
  - without needing to trust all workstations
  - rather all trust a central authentication server
  - Efficiency
- two versions in use: 4 & 5

Kerberos Requirements

- first published report identified its requirements as:
  - security
  - reliability
  - transparency
  - scalability
- implemented using an authentication protocol based on Needham-Schroeder
- A pure private-key scheme

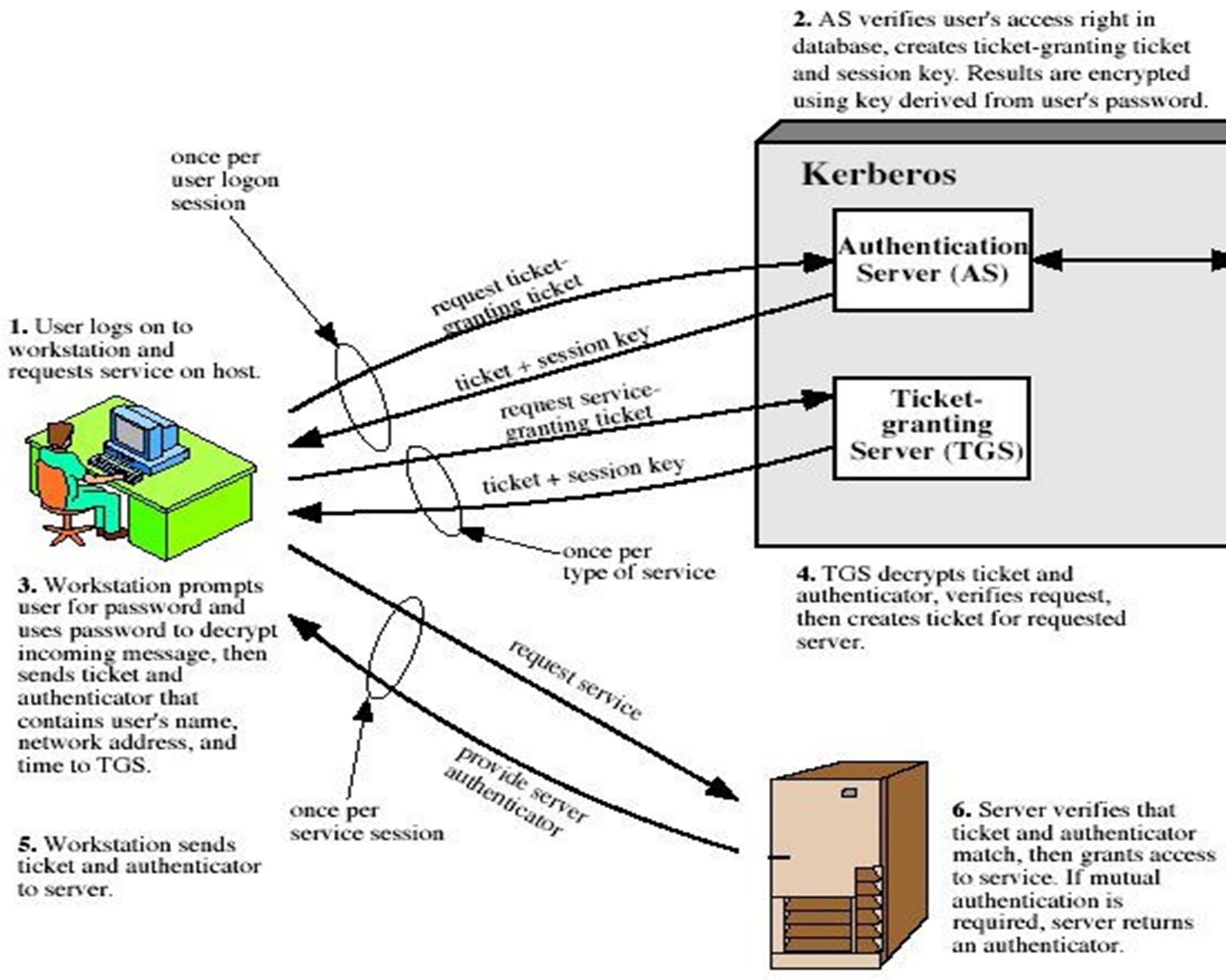
A 3-step improvements leading to Kerberos V4

- A simple authentication dialogue
  - Has to enter password for each server
  - Plaintext transmission of password
- AS+TGS model
  - Enter the password once for multiple services
  - Difficulty in choosing lifetime
- V4 model
  - Use private session keys
  - Can also verify server
  - AS is the KDC for (C, TGS)
  - TGS is the KDC for (C, V)

Kerberos 4 Overview

- a basic third-party authentication scheme
- have an Authentication Server (AS)
  - users initially negotiate with AS to identify self
  - AS provides a authentication credential (ticket granting ticket TGT)
- have a Ticket Granting server (TGS)
  - users subsequently request access to other services from TGS on basis of users TGT

Kerberos 4 Overview


2. AS verifies user's access right in database, creates ticket-granting ticket and session key. Results are encrypted using key derived from user's password.
Kerberos
Authentication Server (AS)
Ticket-granting Server (TGS)
once per user logon session
request ticket-granting ticket
ticket + session key
request service-granting ticket
ticket + session key
1. User logs on to workstation and requests service on host.
once per type of service
3. Workstation prompts user for password and uses password to decrypt incoming message, then sends ticket and authenticator that contains user's name, network address, and time to TGS.
4. TGS decrypts ticket and authenticator, verifies request, then creates ticket for requested server.
request service
provide server authenticator
once per service session
5. Workstation sends ticket and authenticator to server.
6. Server verifies that ticket and authenticator match, then grants access to service. If mutual authentication is required, server returns an authenticator.

Kerberos Realms

- a Kerberos environment consists of:
  - a Kerberos server
  - a number of clients, all registered with server
  - application servers, sharing keys with server
- this is termed a realm
  - typically a single administrative domain
- Inter-realm authentication possible
  - Mutual trust required

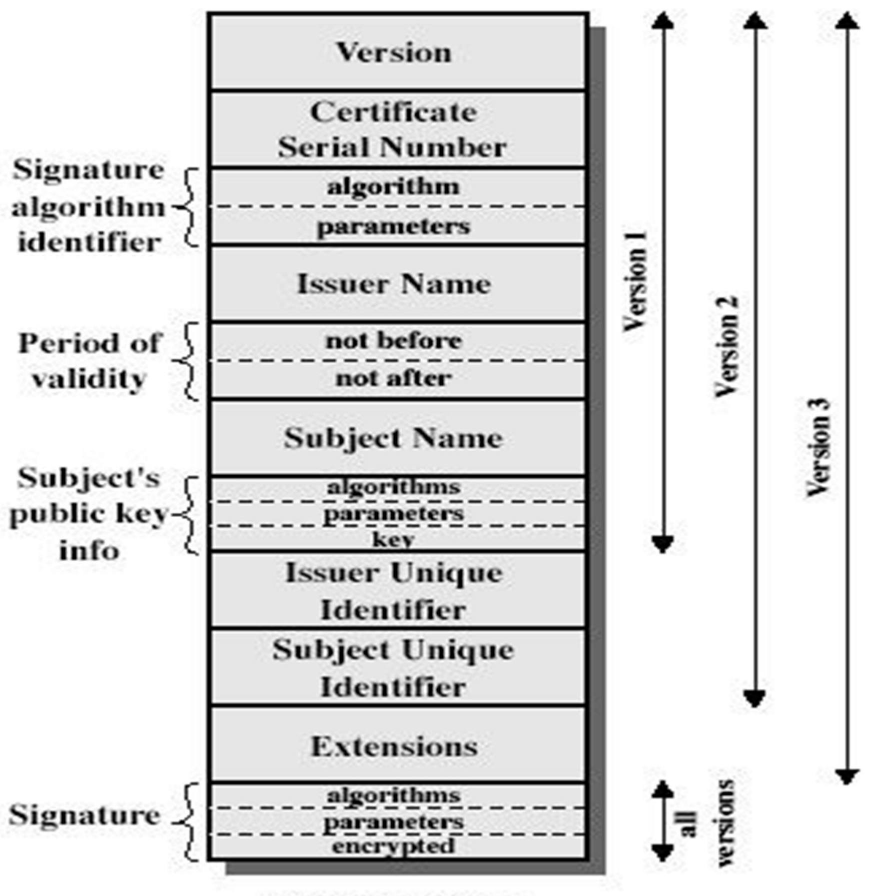
Kerberos Version 5

- developed in mid 1990’s
- provides improvements over v4
  - addresses environmental shortcomings
    - encryption alg, network protocol, byte order, ticket lifetime, authentication forwarding, interrealm auth
  - and technical deficiencies
    - double encryption, non-std mode of use, subsession keys
- specified as Internet standard RFC 1510

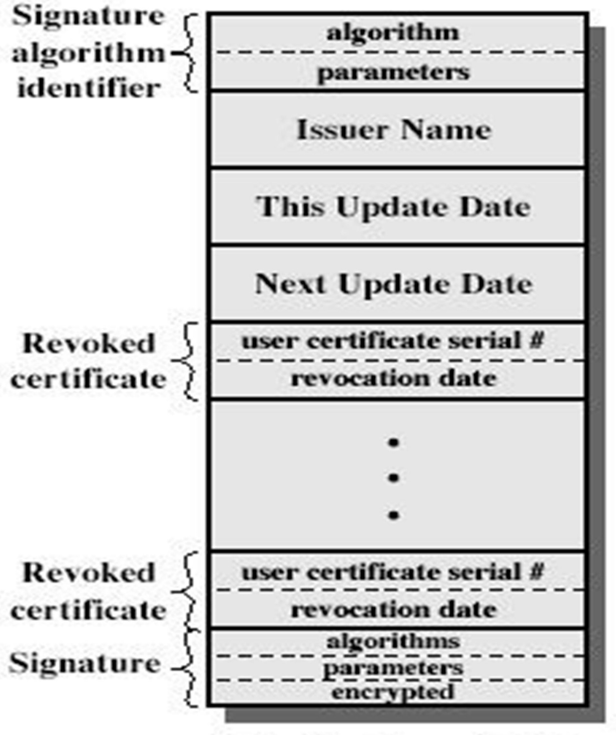
X.509 Authentication Service

- part of CCITT X.500 directory service standards
  - distributed servers maintaining some info database
- defines framework for authentication services
  - directory may store public-key certificates
  - with public key of user
  - signed by certification authority
- also defines authentication protocols
- uses public-key crypto & digital signatures
  - algorithms not standardised, but RSA recommended
  - Used in various contexts, e.g email security, IP security, web security

X.509 Certificates



(a) X.509 Certificate



(b) Certificate Revocation List

X.509 Certificates

- issued by a Certification Authority (CA), containing:
  - version (1, 2, or 3)
  - serial number (unique within CA) identifying certificate
  - signature algorithm identifier
  - issuer X.500 name (CA)
  - period of validity (from - to dates)
  - subject X.500 name (name of owner)
  - subject public-key info (algorithm, parameters, key)
  - issuer unique identifier (v2+) , in case of name reuse
  - subject unique identifier (v2+) , in case of name reuse
  - extension fields (v3)
  - signature (of hash of all fields in certificate, encrypted by the private key of the CA)
- notation CA<<A>> denotes certificate for A signed by CA

Obtaining a Certificate

- any user with access to CA can get any certificate from it
- only the CA can modify a certificate
- because cannot be forged, certificates can be placed in a public directory

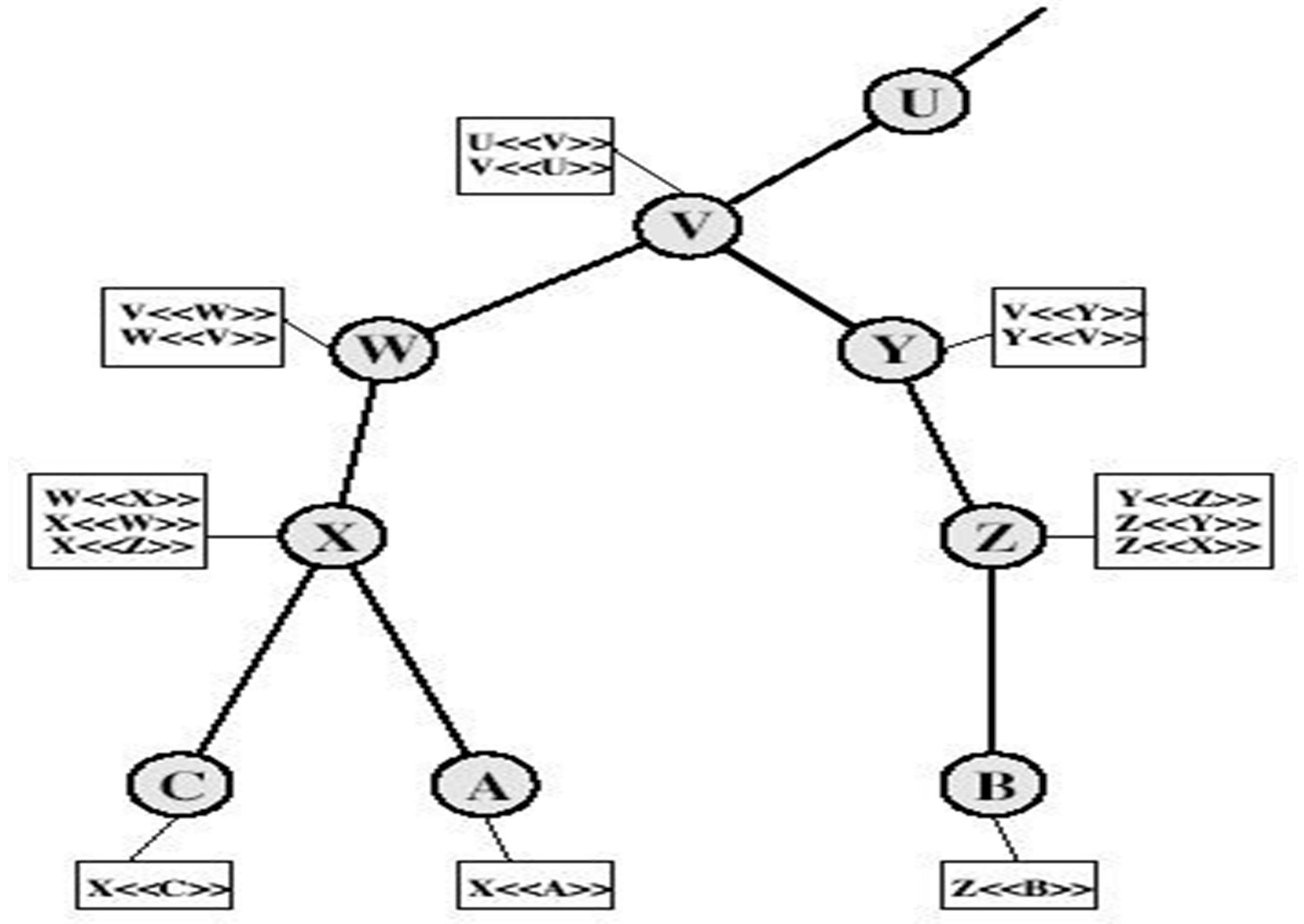
Multiple CAs

- Users in one CA are OK
- What if users from different CAs
  - A from X1
  - B from X2
  - B's certificate is useless to A w/o knowing X2's public key
  - Can work if two CAs exchanged public keys
  - A can use X1<<X2>> , X2<<B>>
- Chain: X1<<X2>> X2<<X3>> … XN<<B>>

CA Hierarchy

- if both users share a common CA then they are assumed to know its public key
- otherwise CA's must form a hierarchy
- use certificates linking members of hierarchy to validate other CA's
  - each CA has certificates for clients (forward) and parent (backward)
- each client trusts parents certificates
- enable verification of any certificate from one CA by users of all other CAs in hierarchy

CA Hierarchy Use



Certificate Revocation

- certificates have a period of validity
- may need to revoke before expiry, eg:
  1. user's private key is compromised
  2. user is no longer certified by this CA
  3. CA's certificate is compromised
- CA’s maintain list of revoked certificates
  – the Certificate Revocation List (CRL)
- users should check certs with CA’s CRL

Authentication Procedures

- X.509 includes three alternative authentication procedures:
  - Assumes each already knows the certified public key of the other
- One-Way Authentication
- Two-Way Authentication
- Three-Way Authentication
- all use public-key signatures

One-Way Authentication

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

Two-Way Authentication

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

Three-Way Authentication

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

X.509 Version 3

- has been recognised that additional information is needed in a certificate
  - email/URL, policy details, usage constraints
- rather than explicitly naming new fields defined a general extension method
- extensions consist of:
  - extension identifier
  - criticality indicator
  - extension value

Certificate Extensions

- key and policy information
  - convey info about subject & issuer keys, plus indicators of certificate policy
- certificate subject and issuer attributes
  - support alternative names, in alternative formats for certificate subject and/or issuer
- certificate path constraints
  - allow constraints on use of certificates by other CA’s

Email Security Enhancements

- confidentiality
  - protection from disclosure
- authentication
  - of sender of message
- message integrity
  - protection from modification
- non-repudiation of origin
  - protection from denial by sender

Pretty Good Privacy (PGP)

- widely used de facto secure email
- developed by Phil Zimmermann
- selected best available crypto algs to use
- integrated into a single program
- available on Unix, PC, Macintosh and Amiga systems
- originally free, now have commercial versions available also

PGP Operation – Authentication

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

PGP Operation – Confidentiality

1. sender generates message and random 128-bit number to be used as session key for this message only
2. message is encrypted, using CAST-128 / IDEA/3DES with session key
3. session key is encrypted using RSA with recipient's public key, then attached to message
4. receiver uses RSA with its private key to decrypt and recover session key
5. session key is used to decrypt message

PGP Operation – Confidentiality & Authentication

- uses both services on same message
  - create signature & attach to message
  - encrypt both message & signature
  - attach RSA encrypted session key

PGP Operation – Compression

- by default PGP compresses message after signing but before encrypting
- uses ZIP compression algorithm

PGP Session Keys

- need a session key for each message
  - of varying sizes: 56-bit DES, 128-bit CAST or IDEA, 168-bit Triple-DES
- uses random inputs taken from previous uses and from keystroke timing of user

PGP Key Rings

- each PGP user has a pair of keyrings:
  - public-key ring contains all the public-keys of other PGP users known to this user, indexed by key ID
  - private-key ring contains the public/private key pair(s) for this user, indexed by key ID & encrypted keyed from a hashed passphrase

PGP Key Management

- rather than relying on certificate authorities
- in PGP every user is own CA
  - can sign keys for users they know directly
- forms a "web of trust"

S/MIME (Secure/Multipurpose Internet Mail Extensions)

- security enhancement to MIME email
  - original Internet RFC822 email was text only
  - MIME provided support for varying content types and multi-part messages
    - Image, video, audio, PS, octet-stream
  - S/MIME added security enhancements
- have S/MIME support in various modern mail agents: MS Outlook, Netscape etc

S/MIME Functions

- enveloped data
  - encrypted content and associated keys
- signed data
  - encoded message + signed digest
- clear-signed data
  - cleartext message + encoded signed digest
- signed & enveloped data
  - nesting of signed & encrypted entities

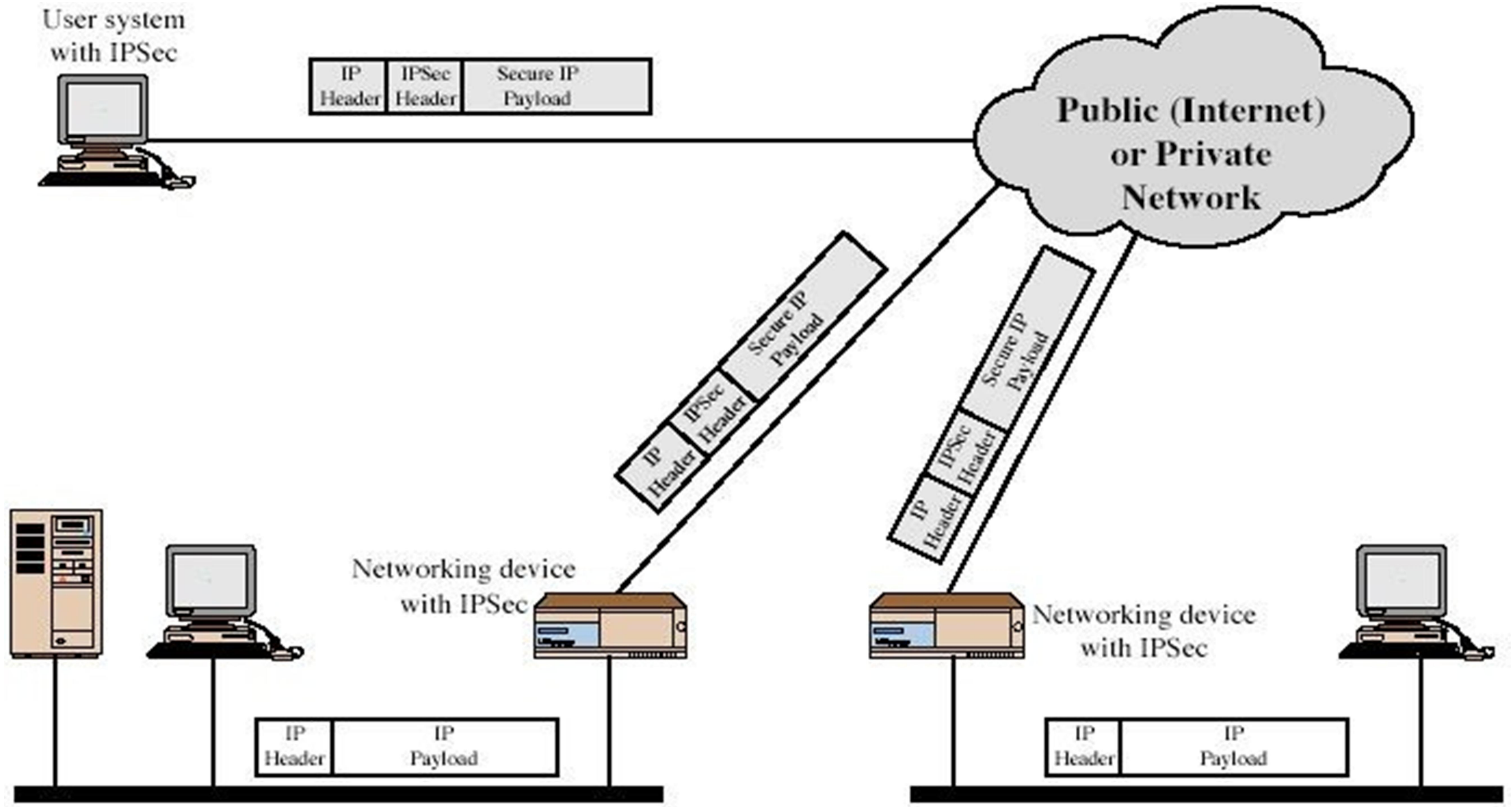
S/MIME Cryptographic Algorithms

- hash functions: SHA-1 & MD5
- digital signatures: DSS & RSA
- session key encryption: D-H & RSA
- message encryption: Triple-DES, RC2/40 and others
- have a procedure to decide which algorithms to use
  - According to the capability of the receiving agent

IP Security

- have considered some application specific security mechanisms
  - eg. S/MIME, PGP, Kerberos, SSL/HTTPS
- however there are security concerns that cut across protocol layers
  - would like security implemented by the network for all applications

IPSec Uses


User system with IPSec
IP Header
IPSec Header
Secure IP Payload
Public (Internet) or Private Network
IP Header
IPSec Header
Secure IP Payload
IP Header
IPSec Header
Secure IP Payload
Networking device with IPSec
Networking device with IPSec
IP Header
IP Payload
IP Header
IP Payload

IPSec

- general IP Security mechanisms
- provides
  - authentication
  - confidentiality
  - key management
- applicable to use over LANs, across public & private WANs, & for the Internet

Benefits of IPSec

- in a firewall/router provides strong security to all traffic crossing the perimeter
- is resistant to bypass
- is below transport layer, hence transparent to applications
- can be transparent to end users

IP Security Architecture

- specification is quite complex
- defined in numerous RFC’s
  - incl. RFC 2401/2402/2406/2408
  - many others, grouped by category
- mandatory in IPv6, optional in IPv4

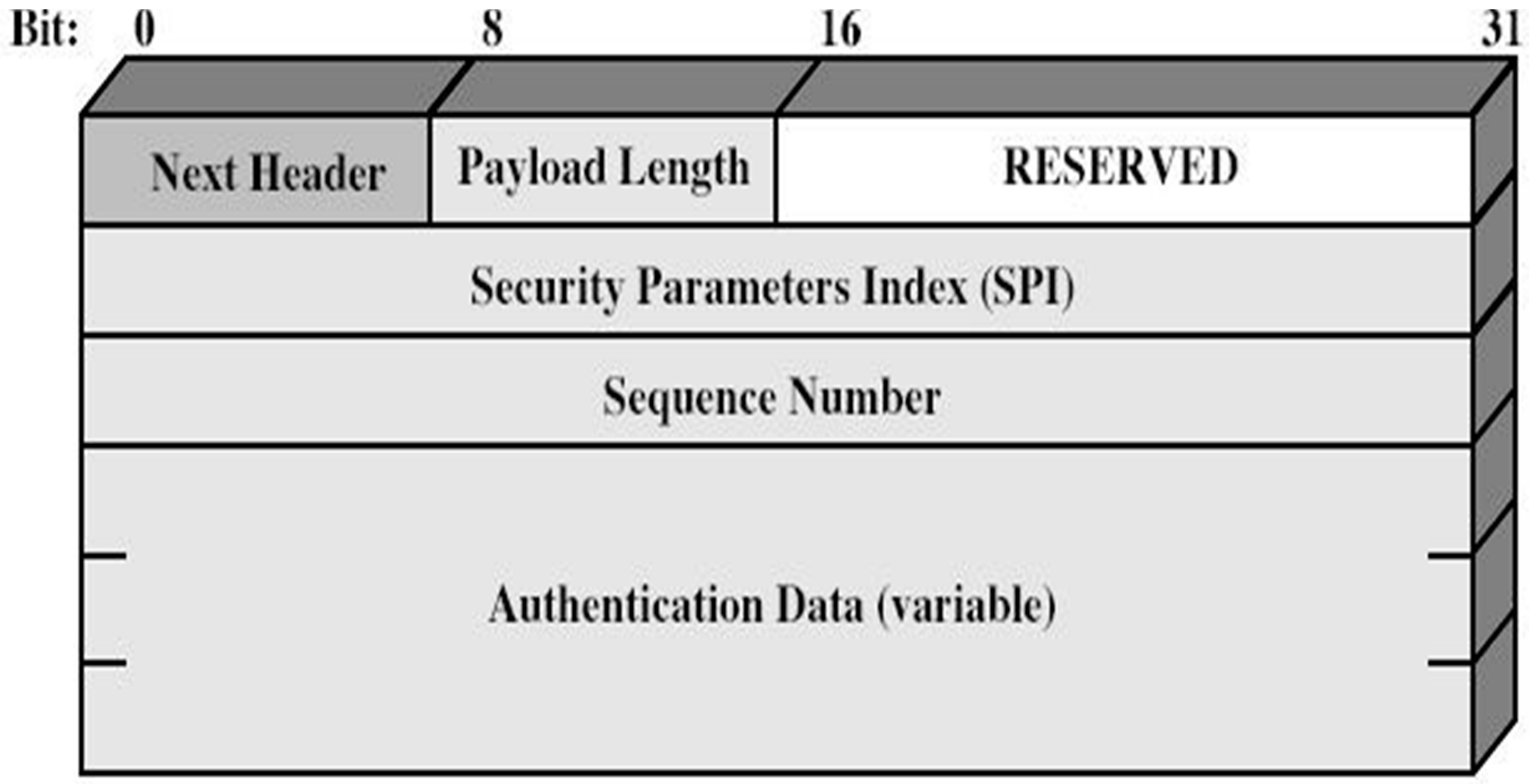
IPSec Protocols

- Authentication Header (AH)
  - Authentication
- Encapsulating Security Payload (ESP)
  - Confidentiality only
  - OR both

Security Associations

- a one-way relationship between sender & receiver that affords security for traffic flow
- defined by 3 parameters:
  – Security Parameters Index (SPI)
  – IP Destination Address
  – Security Protocol Identifier (AH or ESP?)
- has a number of other parameters
  – seq no, AH & EH info, lifetime etc

Authentication Header

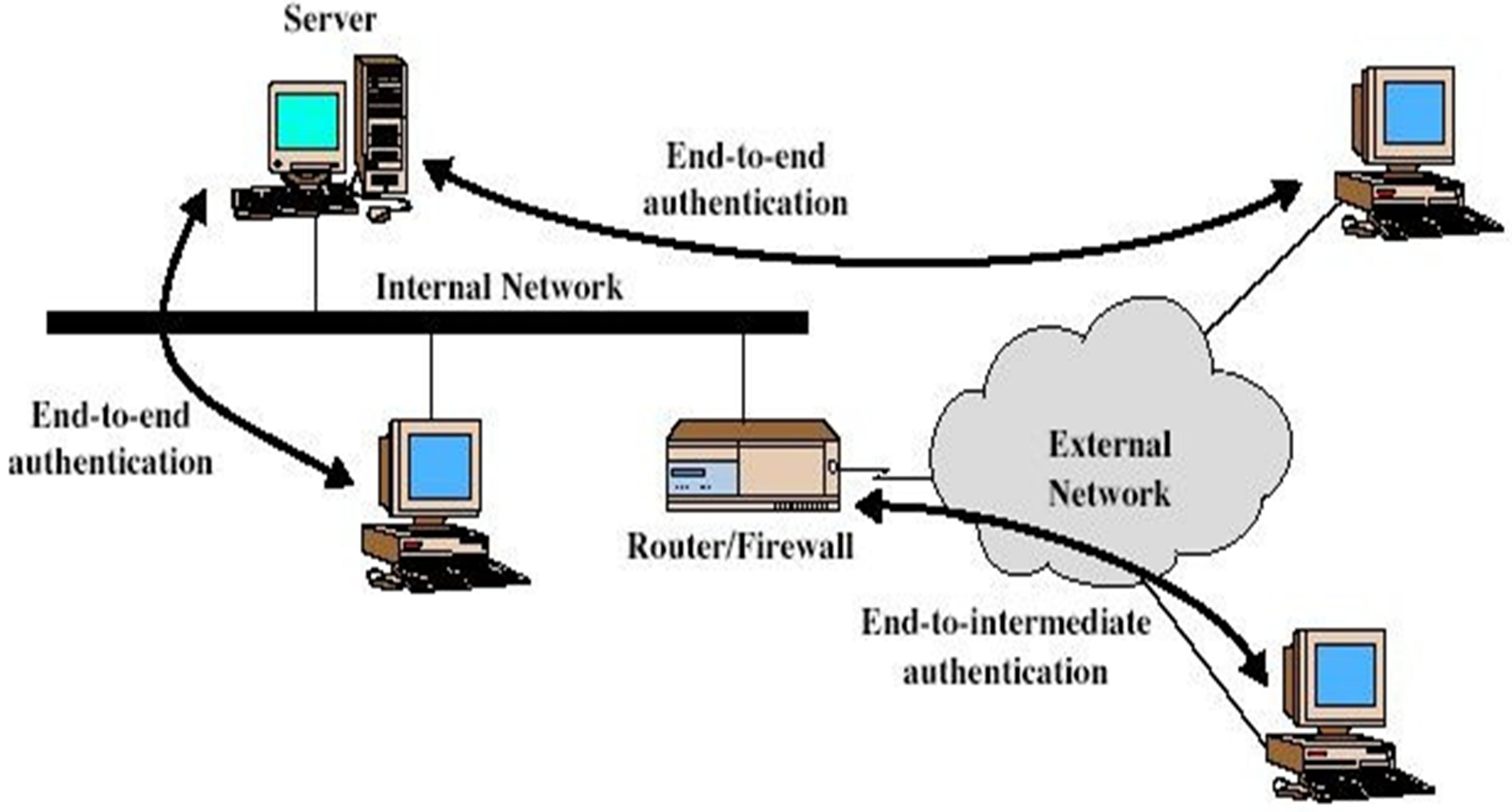


Bit: 0
8
16
31
Next Header
Payload Length
RESERVED
Security Parameters Index (SPI)
Sequence Number
Authentication Data (variable)

Authentication Header (AH)

- provides support for data integrity & authentication of IP packets
  - end system/router can authenticate user/app
  - prevents replay attack by tracking sequence numbers
- based on use of a MAC
  - HMAC-MD5-96 or HMAC-SHA-1-96
- parties must share a secret key

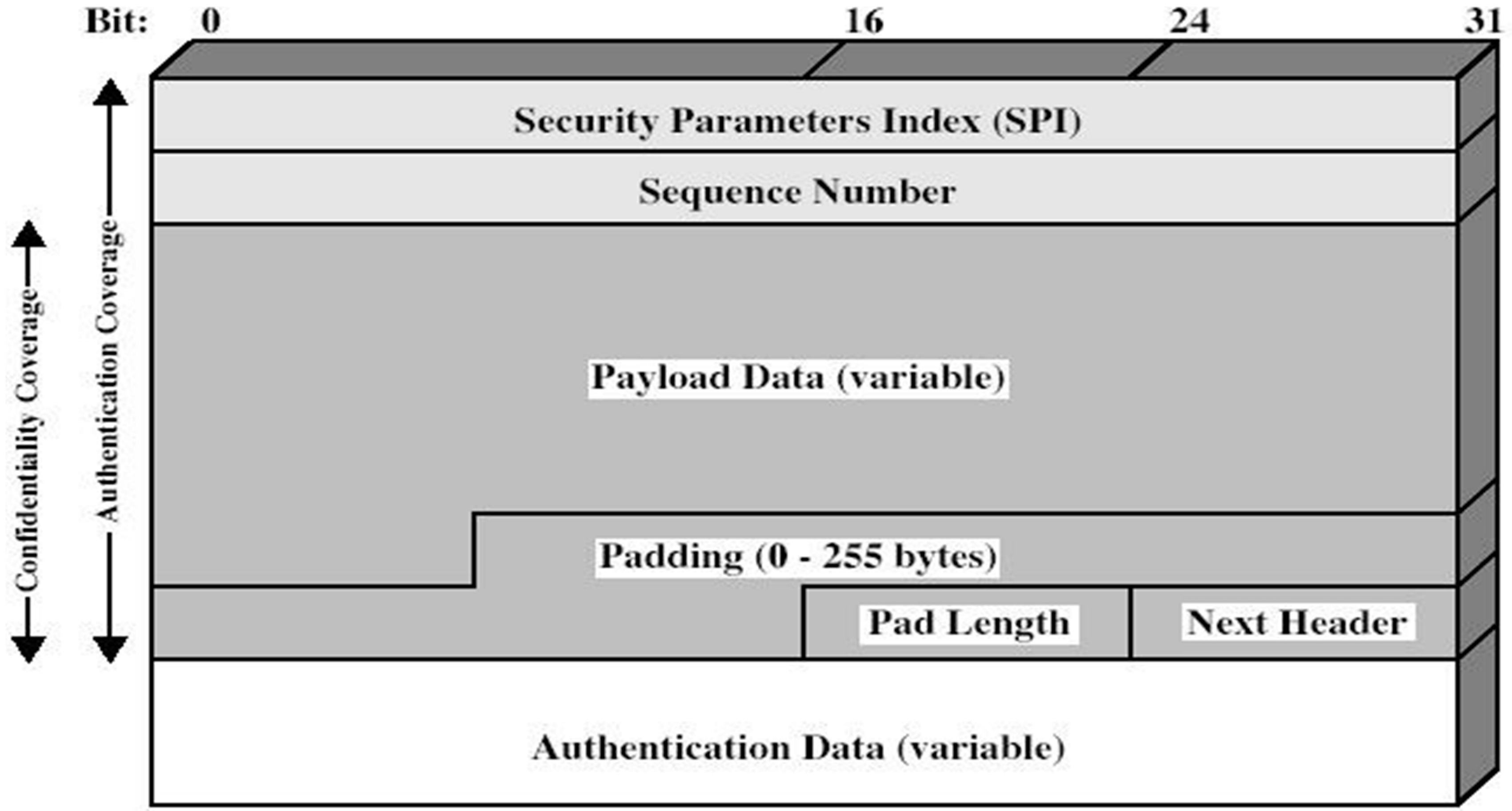
Transport & Tunnel Modes


Server
End-to-end authentication
Internal Network
End-to-end authentication
Router/Firewall
External Network
End-to-intermediate authentication

Encapsulating Security Payload (ESP)

- provides message content confidentiality
- can optionally provide the same authentication services as AH
- supports range of ciphers, modes, padding
  – incl. DES, Triple-DES, RC5, IDEA, CAST etc
  – CBC most common

Encapsulating Security Payload



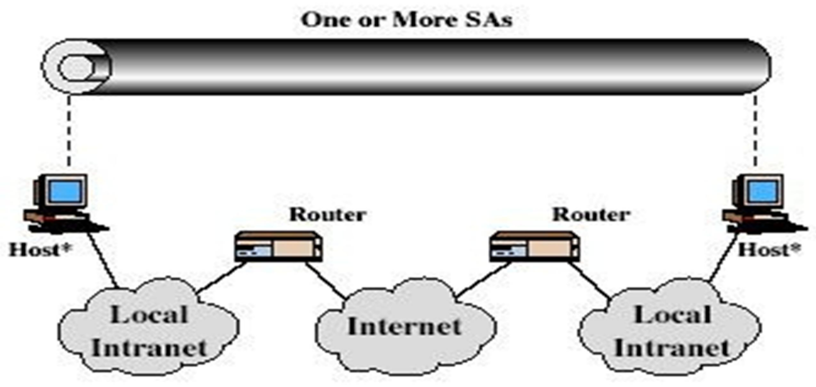
Transport vs Tunnel Mode ESP

- transport mode is used to encrypt & optionally authenticate IP data
  - data protected but header left in clear
  - can do traffic analysis but is efficient
  - good for ESP host to host traffic
- tunnel mode encrypts entire IP packet
  - add new header for next hop
  - good for VPNs, gateway to gateway security

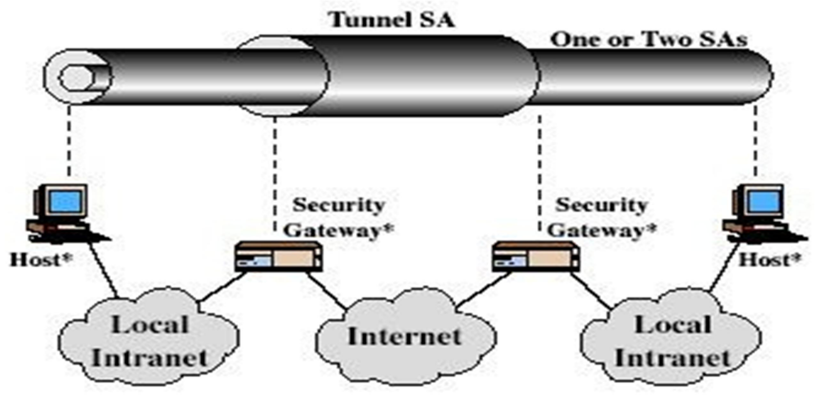
Combining Security Associations

- SA’s can implement either AH or ESP
- to implement both need to combine SA’s
  - form a security bundle

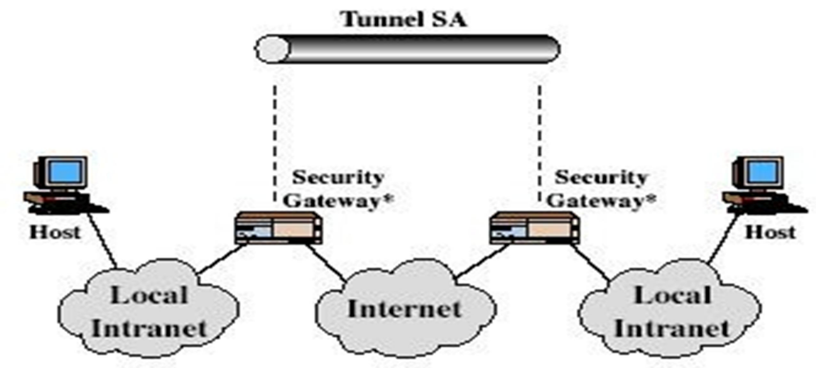
Combining Security Associations



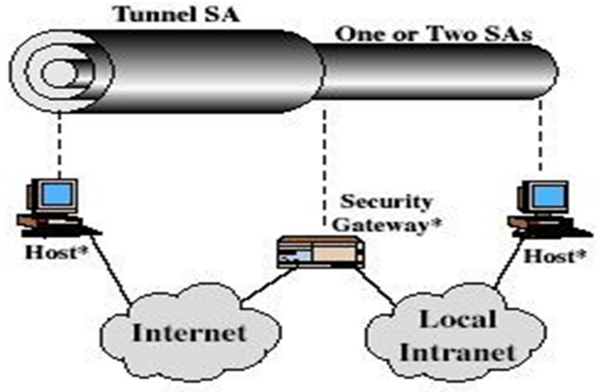
(a) Case 1



(c) Case 3

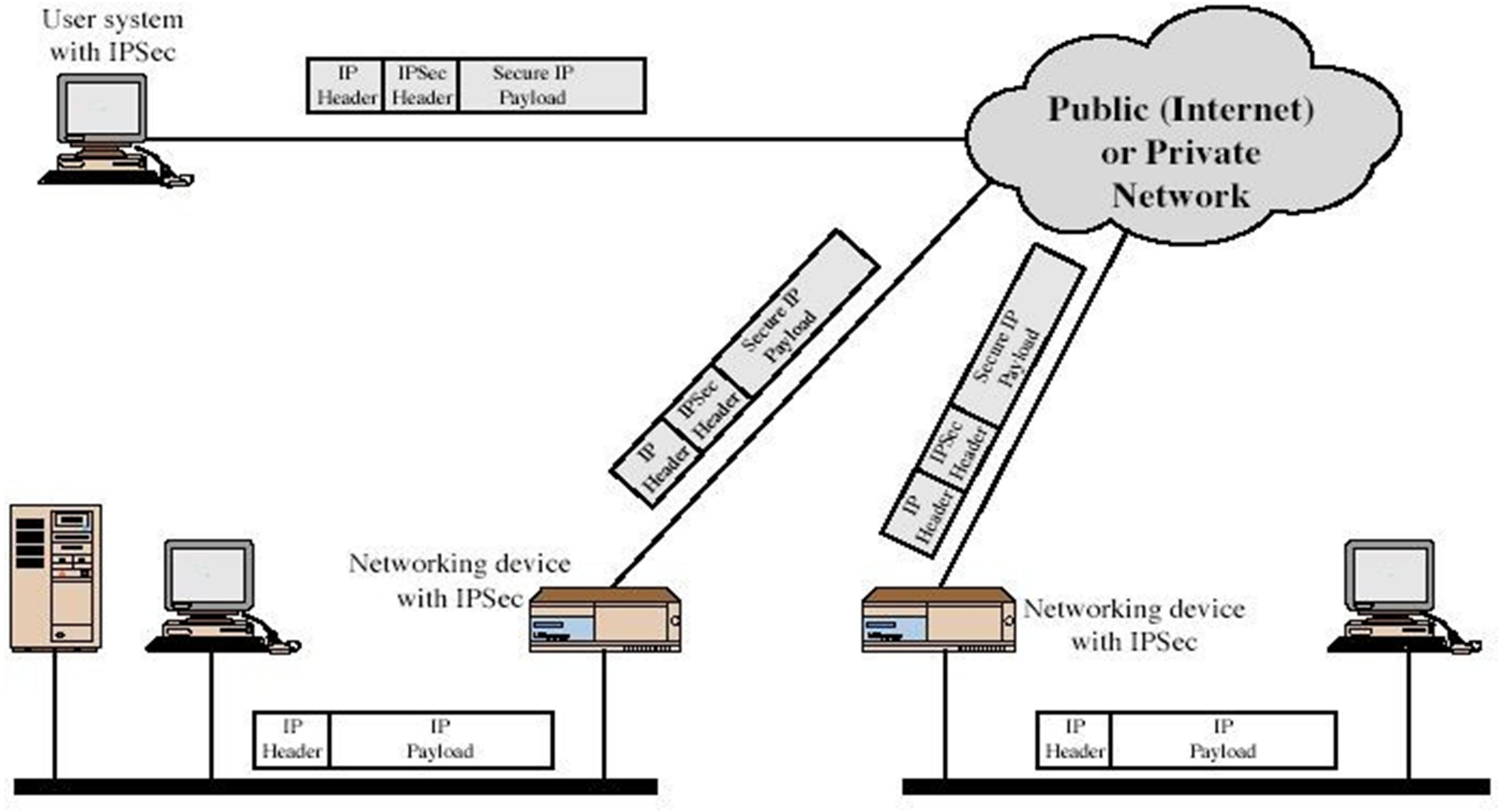


(b) Case 2



(d) Case 4

IPSec Uses


User system with IPSec
IP Header
IPSec Header
Secure IP Payload
Public (Internet) or Private Network
IP Header
IPSec Header
Secure IP Payload
IP Header
IPSec Header
Secure IP Payload
Networking device with IPSec
Networking device with IPSec
IP Header
IP Payload
IP Header
IP Payload

IPSec

- general IP Security mechanisms
- provides
  - authentication
  - confidentiality
  - key management
- applicable to use over LANs, across public & private WANs, & for the Internet

Benefits of IPSec

- in a firewall/router provides strong security to all traffic crossing the perimeter
- is resistant to bypass
- is below transport layer, hence transparent to applications
- can be transparent to end users

IP Security Architecture

- specification is quite complex
- defined in numerous RFC’s
  - incl. RFC 2401/2402/2406/2408
  - many others, grouped by category
- mandatory in IPv6, optional in IPv4

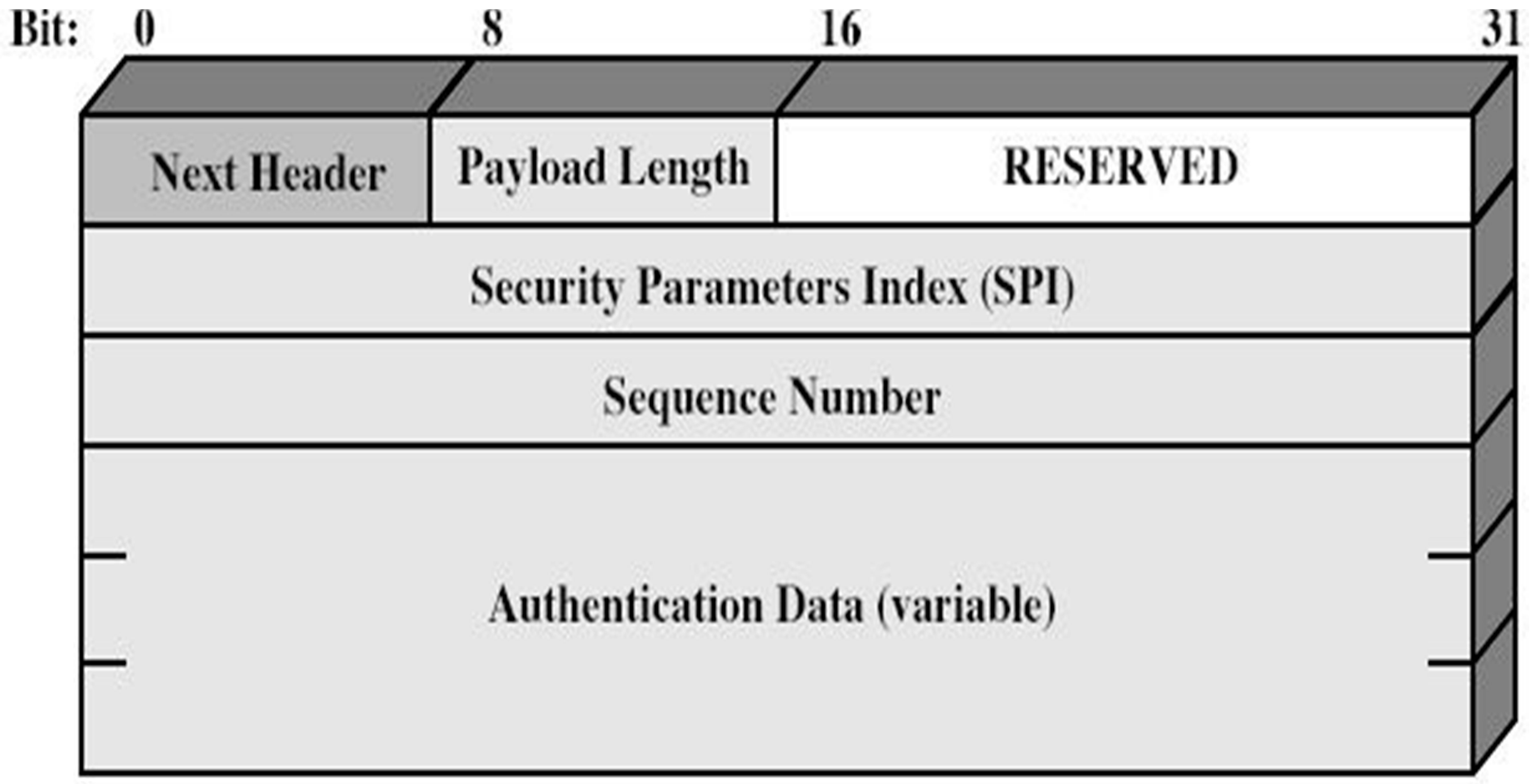
IPSec Protocols

- Authentication Header (AH)
  - Authentication
- Encapsulating Security Payload (ESP)
  - Confidentiality only
  - OR both

Security Associations

- a one-way relationship between sender & receiver that affords security for traffic flow
- defined by 3 parameters:
  - Security Parameters Index (SPI)
  - IP Destination Address
  - Security Protocol Identifier (AH or ESP?)
- has a number of other parameters
  - seq no, AH & EH info, lifetime etc

Authentication Header

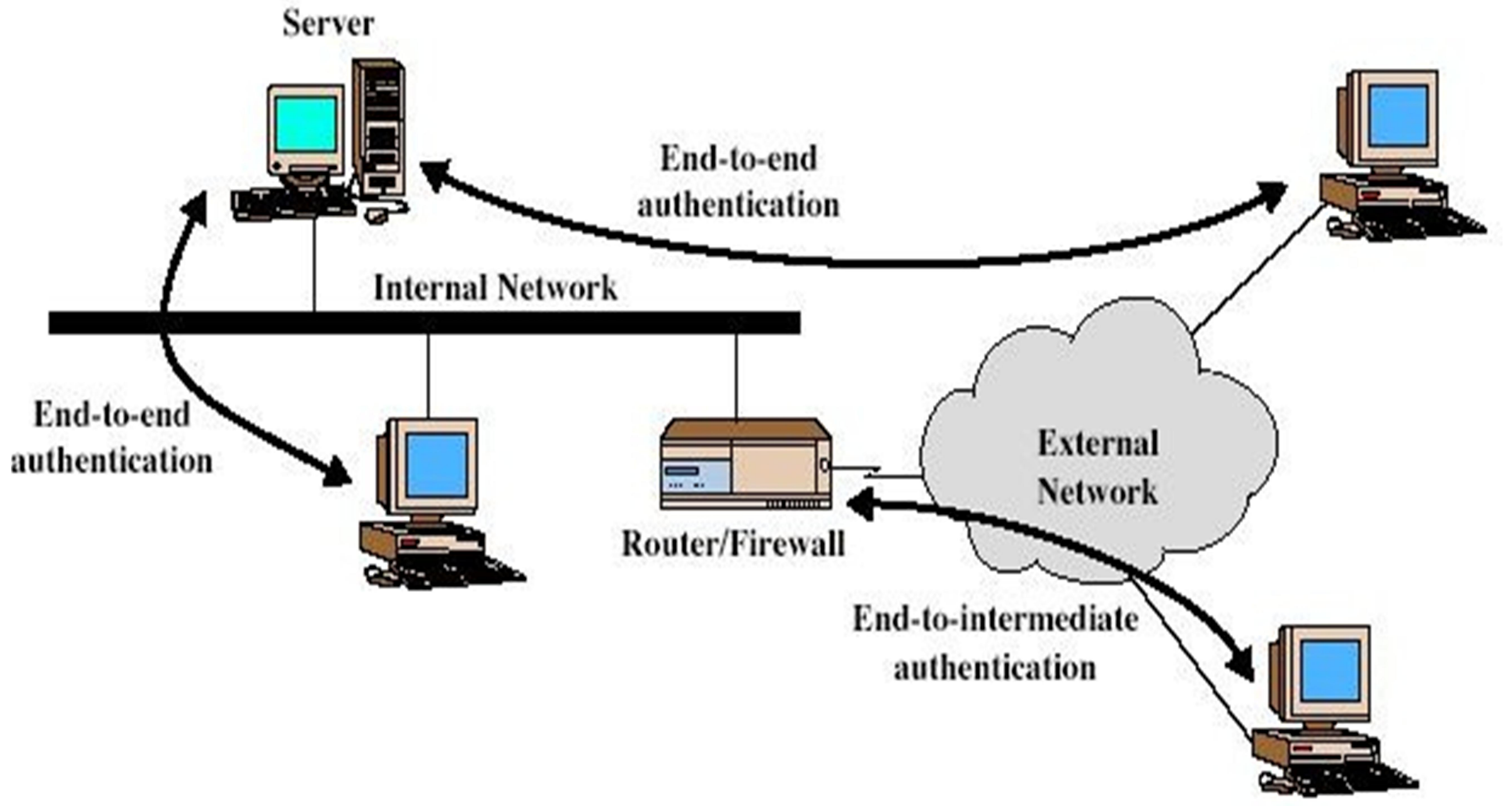


Bit: 0
8
16
31
Next Header
Payload Length
RESERVED
Security Parameters Index (SPI)
Sequence Number
Authentication Data (variable)

Authentication Header (AH)

- provides support for data integrity & authentication of IP packets
  - end system/router can authenticate user/app
  - prevents replay attack by tracking sequence numbers
- based on use of a MAC
  - HMAC-MD5-96 or HMAC-SHA-1-96

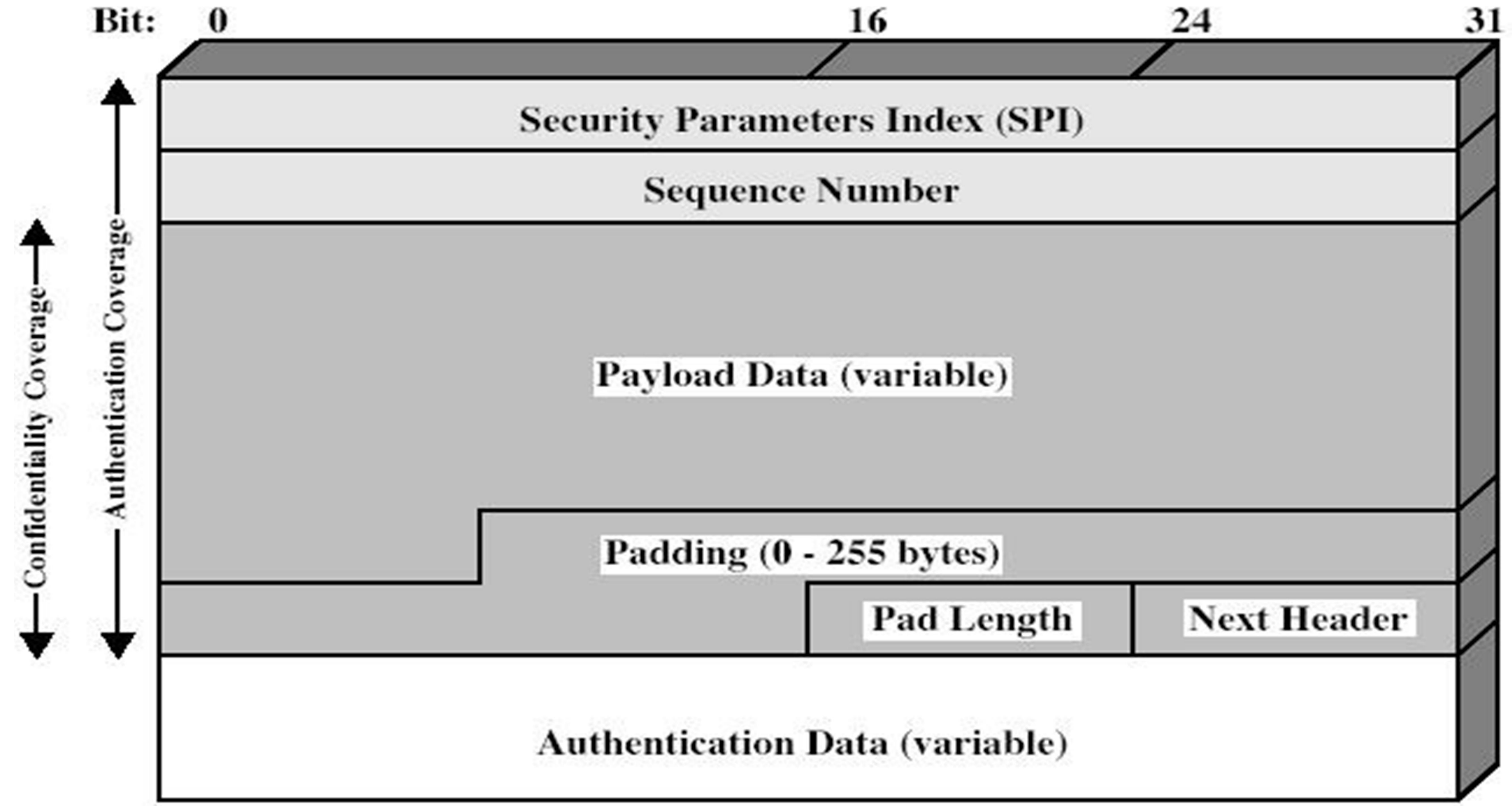
Transport & Tunnel Modes


Server
End-to-end authentication
Internal Network
End-to-end authentication
Router/Firewall
External Network
End-to-intermediate authentication

Encapsulating Security Payload (ESP)

- provides message content confidentiality
- can optionally provide the same authentication services as AH
- supports range of ciphers, modes, padding
  - incl. DES, Triple-DES, RC5, IDEA, CAST etc
  - CBC most common

Encapsulating Security Payload



Transport vs Tunnel Mode ESP

- transport mode is used to encrypt & optionally authenticate IP data
  - data protected but header left in clear
  - can do traffic analysis but is efficient
  - good for ESP host to host traffic
- tunnel mode encrypts entire IP packet
  - add new header for next hop

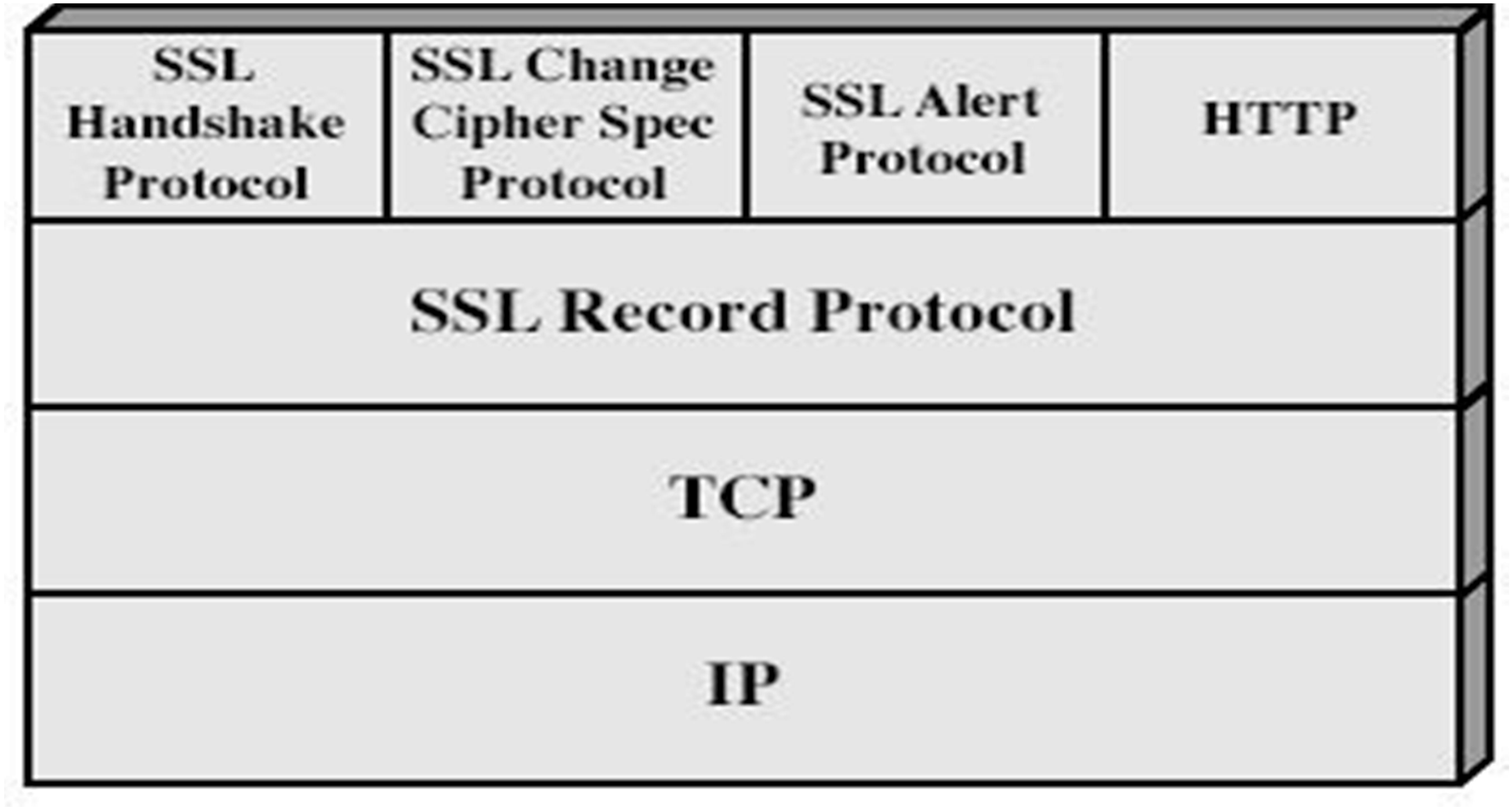
Web Security

- Web now widely used by business, government, individuals
- but Internet & Web are vulnerable
- have a variety of threats
  - integrity
  - confidentiality
  - denial of service
  - authentication
- need added security mechanisms

SSL (Secure Socket Layer)

- Transport layer security service
- Originally developed by Netscape
- Version 3 designed with public input
- Subsequently became Internet standard known as TLS (Transport Layer Security)
- Uses TCP to provide a reliable end-to-end service
- SSL has two layers of protocols

SSL Architecture



SSL Architecture

- **SSL session**
  - an association between client & server
  - created by the Handshake Protocol
  - define a set of cryptographic parameters
  - may be shared by multiple SSL connections
- **SSL connection**
  - a transient, peer-to-peer, communications link
  - associated with 1 SSL session

SSL Record Protocol

- **confidentiality**
  - using symmetric encryption with a shared secret key defined by Handshake Protocol
  - IDEA, RC2-40, DES-40, DES, 3DES, Fortezza, RC4-40, RC4-128
  - message is compressed before encryption
- **message integrity**
  - using a MAC with shared secret key
  - similar to HMAC but with different padding

SSL Change Cipher Spec Protocol

- one of 3 SSL specific protocols which use the SSL Record protocol
- a single message
- causes pending state to become current
- hence updating the cipher suite in use

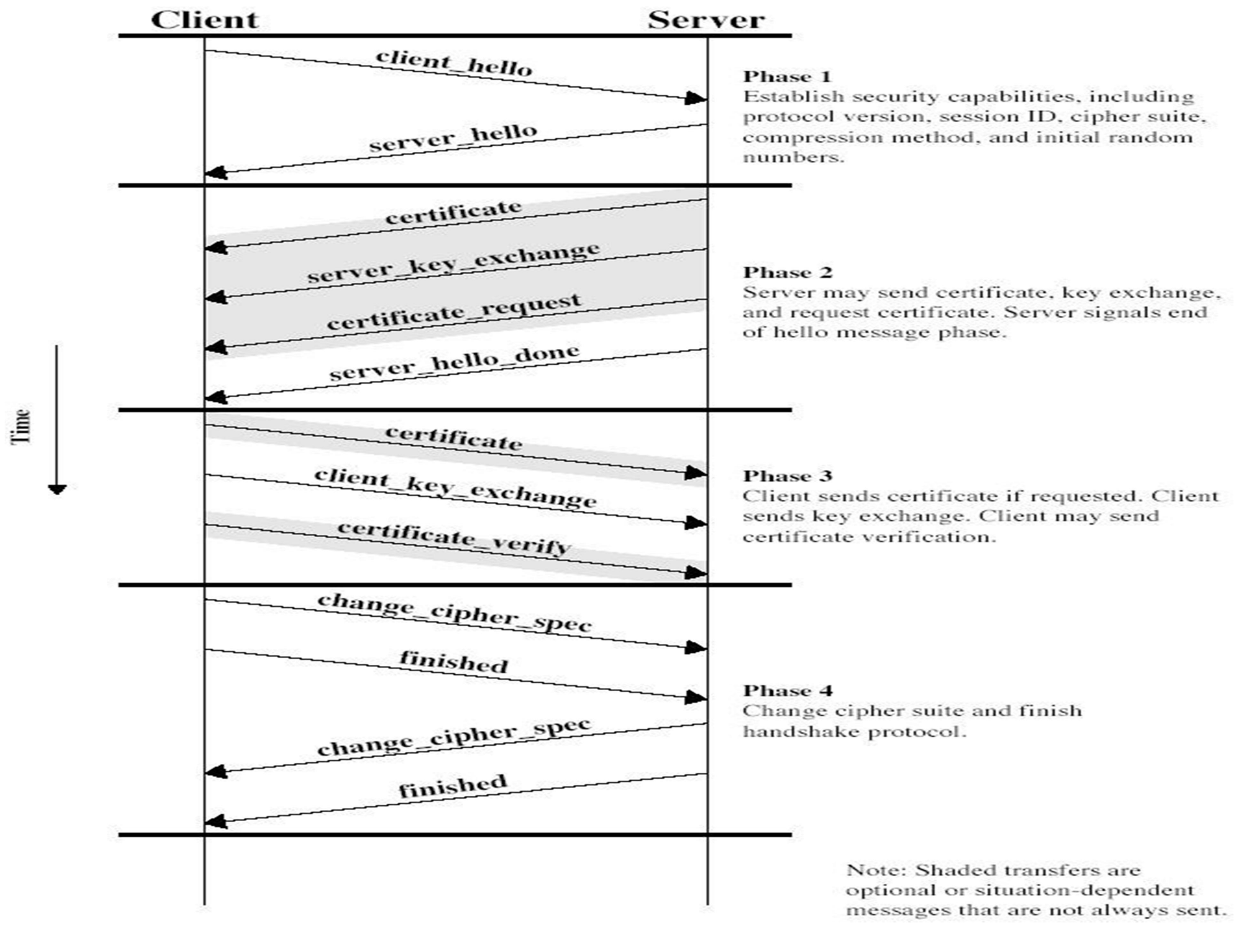
SSL Alert Protocol

- conveys SSL-related alerts to peer entity
- severity
    - warning or fatal
- specific alert
    - unexpected message, bad record mac, decompression failure, handshake failure, illegal parameter
    - close notify, no certificate, bad certificate, unsupported certificate, certificate revoked, certificate expired, certificate unknown
- compressed & encrypted like all SSL data

SSL Handshake Protocol

- allows server & client to:
  - authenticate each other
  - to negotiate encryption & MAC algorithms
  - to negotiate cryptographic keys to be used
- comprises a series of messages in phases
  - Establish Security Capabilities
  - Server Authentication and Key Exchange
  - Client Authentication and Key Exchange
  - Finish

SSL Handshake Protocol



Client
Server
client_hello
server_hello
Phase 1
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
Phase 2
Server may send certificate, key exchange, and request certificate. Server signals end of hello message phase.
Time
certificate
client_key_exchange
certificate_verify
Phase 3
Client sends certificate if requested. Client sends key exchange. Client may send certificate verification.
change_cipher_spec
finished
change_cipher_spec
finished
Phase 4
Change cipher suite and finish handshake protocol.
Note: Shaded transfers are optional or situation-dependent messages that are not always sent.

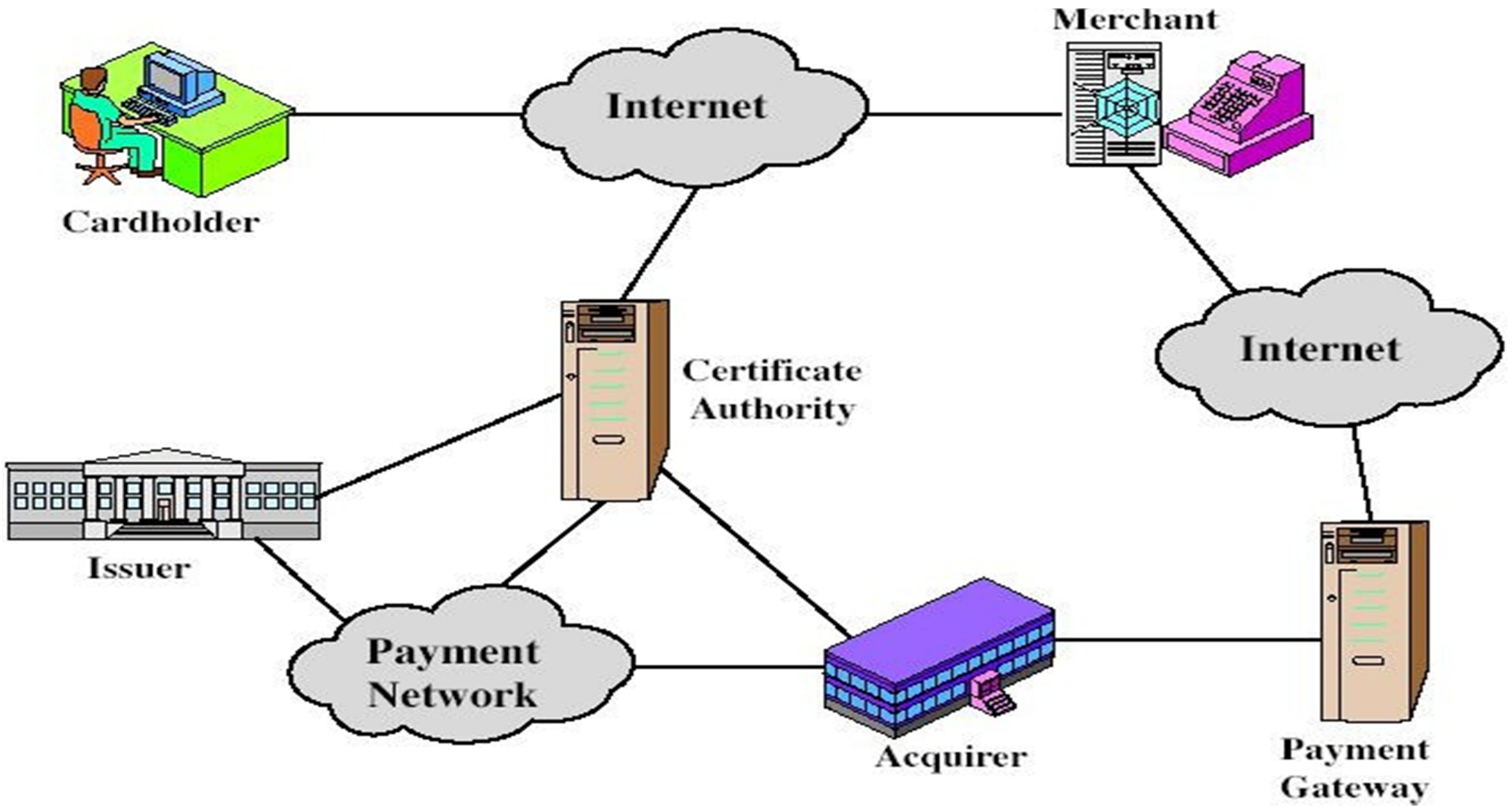
TLS (Transport Layer Security)

- IETF standard RFC 2246 similar to SSLv3
- with minor differences
  - in record format version number
  - uses HMAC for MAC
  - a pseudo-random function expands secrets
  - has additional alert codes
  - some changes in supported ciphers
  - changes in certificate negotiations

Secure Electronic Transactions (SET)

- open encryption & security specification
- to protect Internet credit card transactions
- developed in 1996 by Mastercard, Visa etc
- not a payment system
- rather a set of security protocols & formats
  - secure communications amongst parties
  - trust from use of X.509v3 certificates
  - privacy by restricted info to those who need it

SET Components



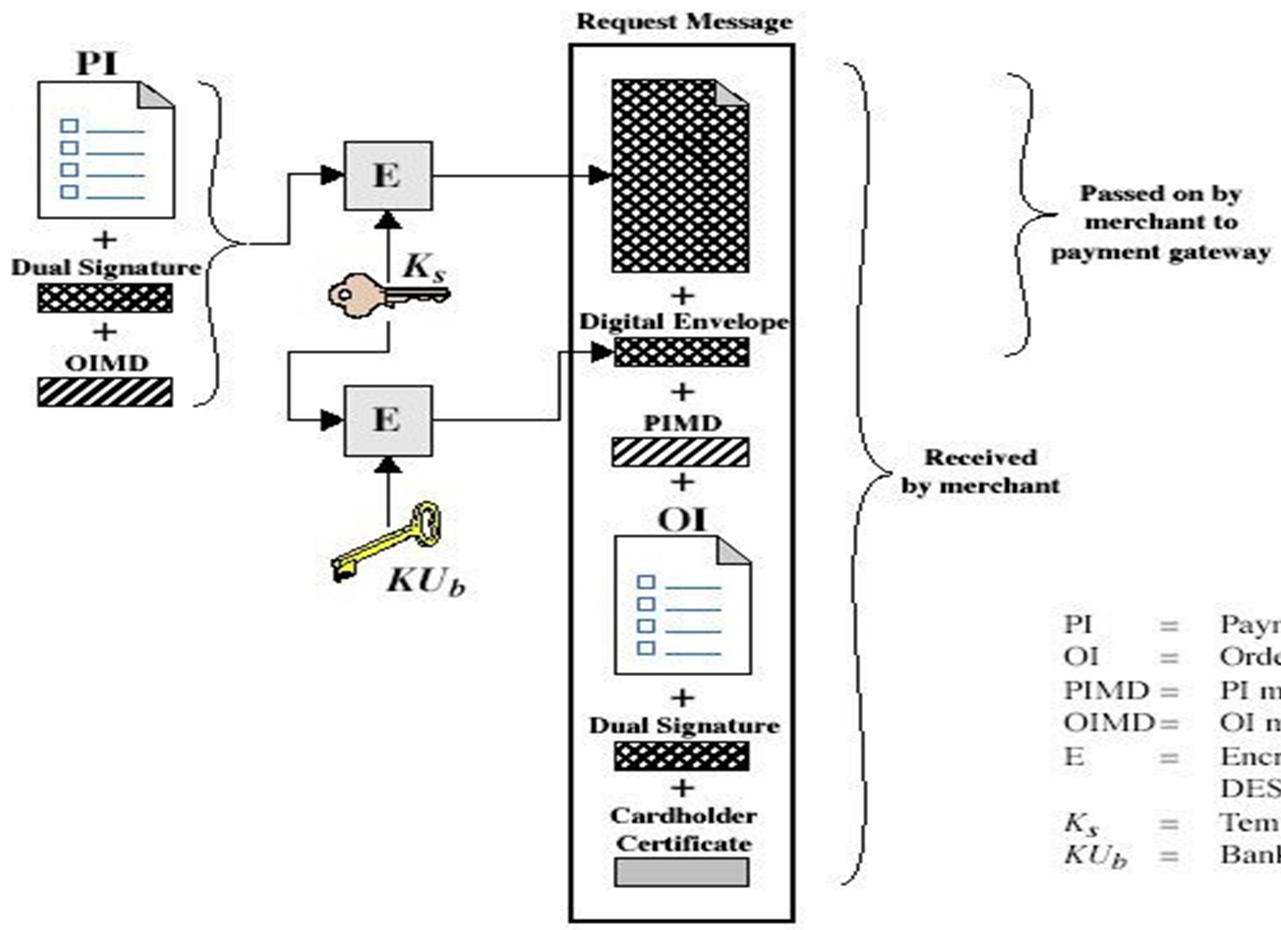
SET Transaction

1. customer opens account
2. customer receives a certificate
3. merchants have their own certificates
4. customer places an order
5. merchant is verified
6. order and payment are sent
7. merchant requests payment authorization
8. merchant confirms order
9. merchant provides goods or service
10. merchant requests payment

Dual Signature

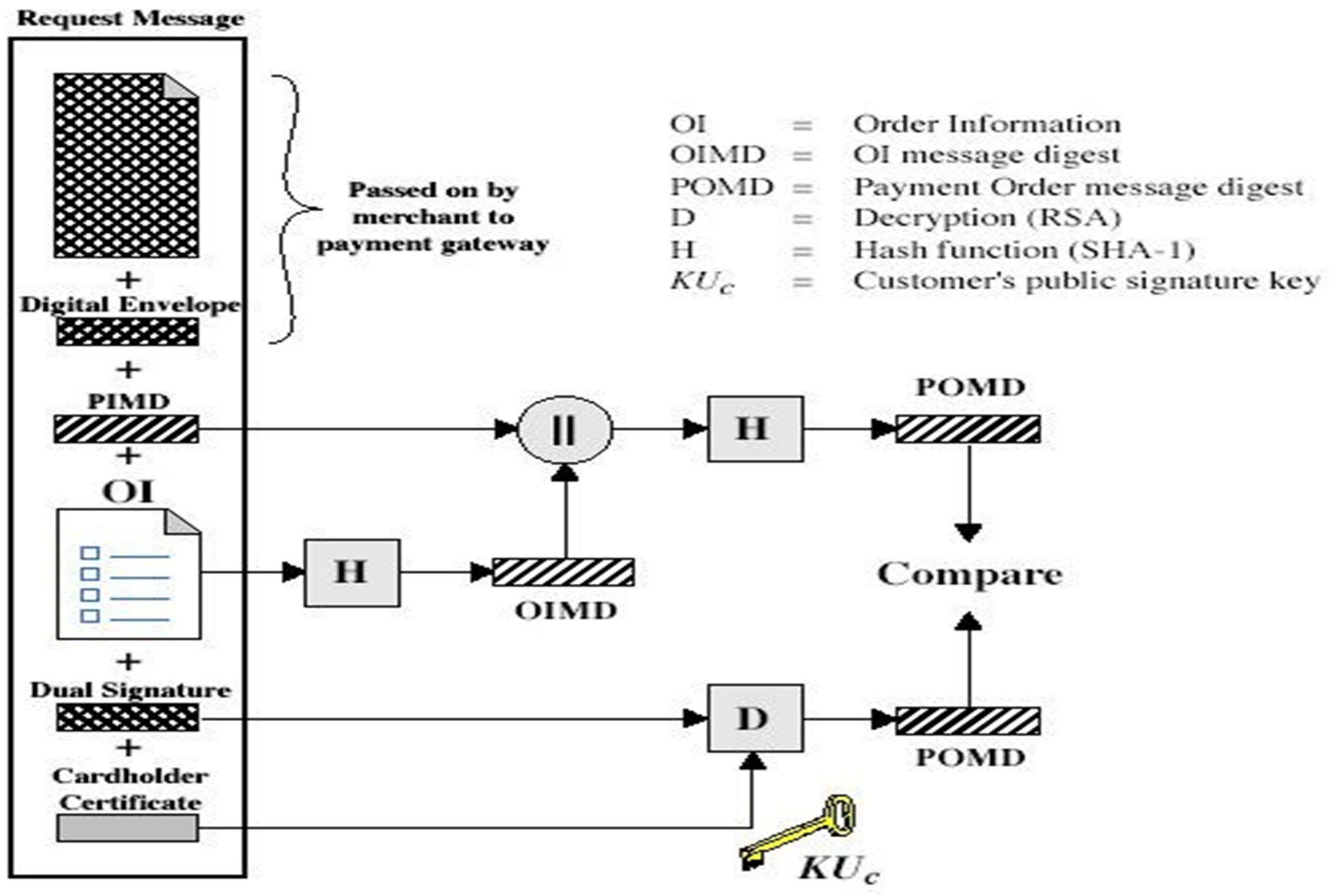
- customer creates dual messages
  - order information (OI) for merchant
  - payment information (PI) for bank
- neither party needs details of other
- but **must** know they are linked
- use a dual signature for this
  - signed concatenated hashes of OI & PI

Purchase Request - Customer



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

Purchase Request – Merchant



Purchase Request – Merchant

1. verifies cardholder certificates using CA sigs
2. verifies dual signature using customer's public signature key to ensure order has not been tampered with in transit & that it was signed using cardholder's private signature key
3. processes order and forwards the payment information to the payment gateway for authorization (described later)
4. sends a purchase response to cardholder

Payment Gateway Authorization

1. verifies all certificates
2. decrypts digital envelope of authorization block to obtain symmetric key & then decrypts authorization block
3. verifies merchant's signature on authorization block
4. decrypts digital envelope of payment block to obtain symmetric key & then decrypts payment block
5. verifies dual signature on payment block
6. verifies that transaction ID received from merchant matches that in PI received (indirectly) from customer
7. requests & receives an authorization from issuer
8. sends authorization response back to merchant

Payment Capture

- merchant sends payment gateway a payment capture request
- gateway checks request
- then causes funds to be transferred to merchants account
- notifies merchant using capture response

Intruders

- significant issue for networked systems is hostile or unwanted access
- either via network or local
- can identify classes of intruders:
  - masquerader
  - misfeasor
  - clandestine user
- varying levels of competence
  - key goal often is to acquire passwords

Password Guessing

- one of the most common attacks
- attacker knows a login (from email/web page etc)
- then attempts to guess password for it
  - try default passwords shipped with systems
  - try all short passwords
  - then try by searching dictionaries of common words
  - intelligent searches try passwords associated with the user (variations on names, birthday, phone, common words/interests)
  - before exhaustively searching all possible passwords
- success depends on password chosen by user
- surveys show many users choose poorly

Password Capture

- another attack involves **password capture**
  - watching over shoulder as password is entered
  - using a Trojan horse program to collect
  - monitoring an insecure network login (eg. telnet, FTP, web, email)
  - extracting recorded info after successful login (web history/cache, last number dialed etc)

Intrusion Detection

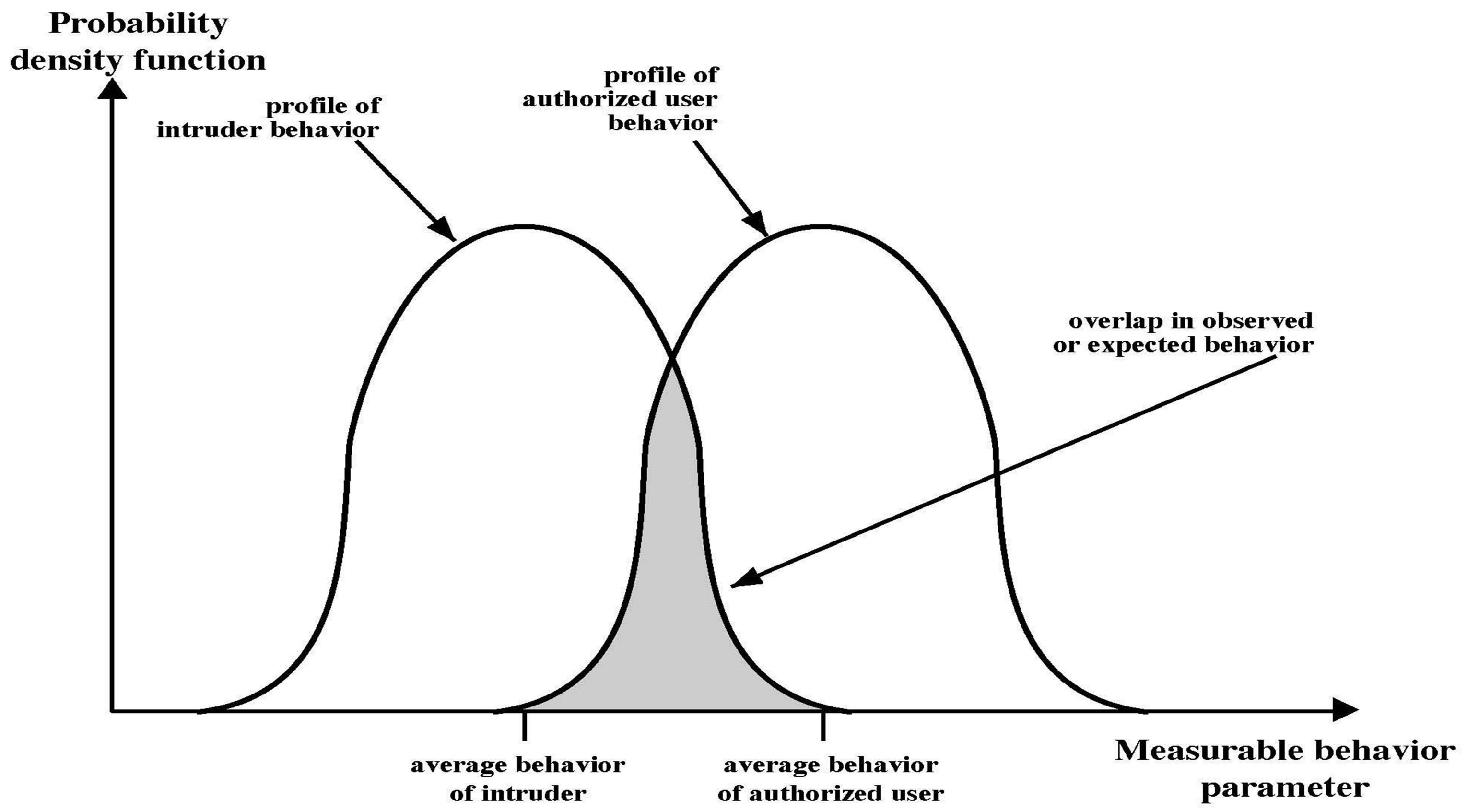

Probability density function
profile of intruder behavior
profile of authorized user behavior
overlap in observed or expected behavior
average behavior of intruder
average behavior of authorized user
Measurable behavior parameter

Figure 18.1 Profiles of Behavior of Intruders and Authorized Users

Approaches to Intrusion Detection

- statistical anomaly detection
  - threshold
  - profile based
- rule-based detection
  - Anomaly, based on previous usage pattern
  - penetration identification

Audit Records

- fundamental tool for intrusion detection
- native audit records
  - part of all common multi-user O/S
- detection-specific audit records
  - created specifically to collect wanted info

Statistical Anomaly Detection

- threshold detection
  - count occurrences of specific event over time
  - if exceed reasonable value assume intrusion
  - alone is a crude & ineffective detector
- profile based
  - characterize past behavior of users
  - detect significant deviations from this
  - profile usually multi-parameter

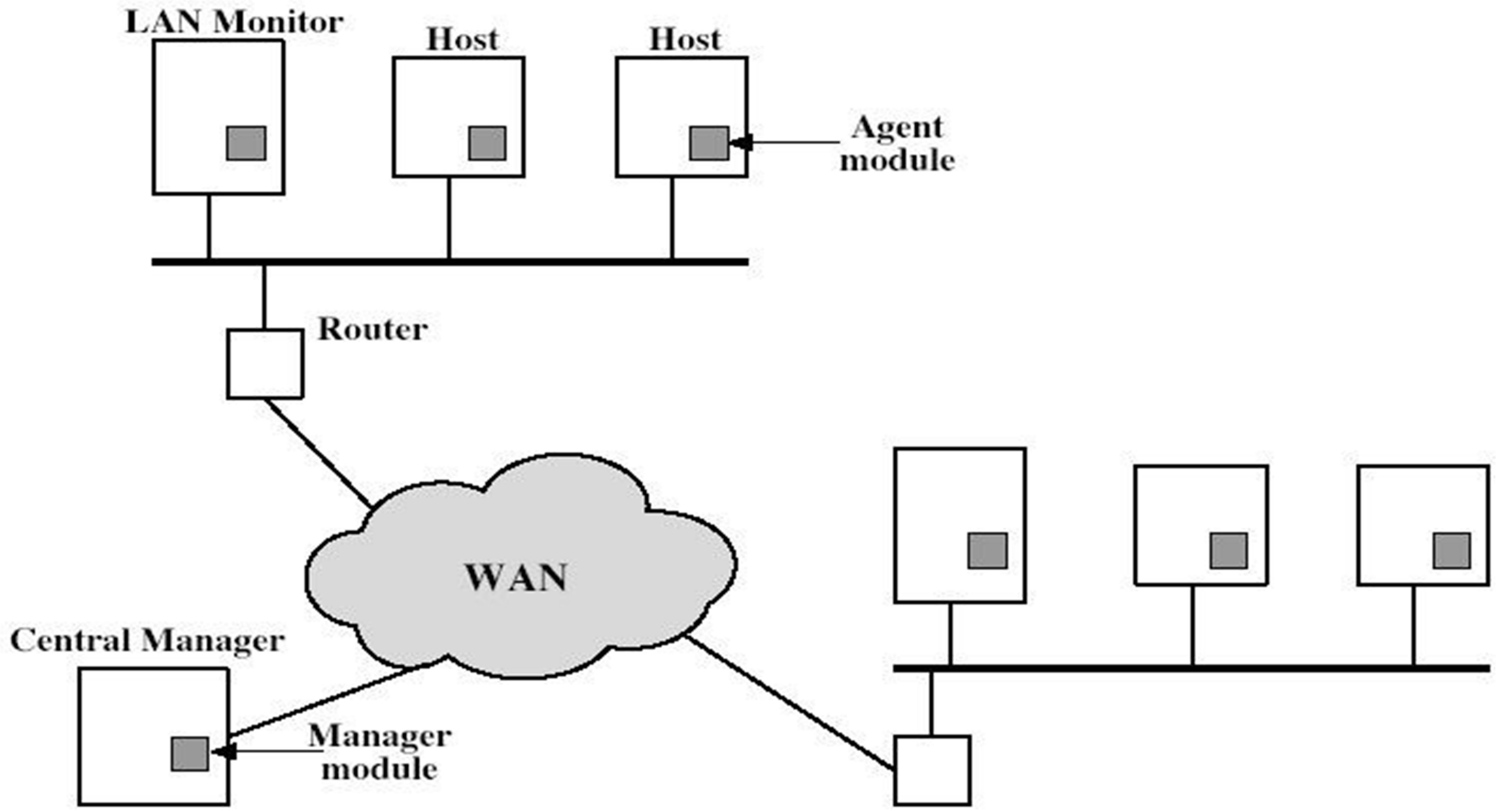
Audit Record Analysis

- foundation of statistical approaches
- analyze records to get metrics over time
  - counter, gauge, interval timer, resource use
- use various tests on these to determine if current behavior is acceptable
  - mean & standard deviation, multivariate, markov process, time series, operational

Table 18.1 Measures That May Be Used for Intrusion Detection

Measure	Model	Type of Intrusion Detected
Login and Session Activity		
Login frequency by day and time	Mean and standard deviation	Intruders may be likely to log in during off-hours.
Frequency of login at different locations	Mean and standard deviation	Intruders may log in from a location that a particular user rarely or never uses.
Time since last login	Operational	Break-in on a "dead" account.
Elapsed time per session	Mean and standard deviation	Significant deviations might indicate masquerader.
Quantity of output to location	Mean and standard deviation	Excessive amounts of data transmitted to remote locations could signify leakage of sensitive data.
Session resource utilization	Mean and standard deviation	Unusual processor or I/O levels could signal an intruder.
Password failures at login	Operational	Attempted break-in by password guessing.
Failures to login from specified terminals	Operational	Attempted break-in.
Command or Program Execution Activity		
Execution frequency	Mean and standard deviation	May detect intruders, who are likely to use different commands, or a successful penetration by a legitimate user, who has gained access to privileged commands.
Program resource utilization	Mean and standard deviation	An abnormal value might suggest injection of a virus or Trojan horse, which performs side-effects that increase I/O or processor utilization.
Execution denials	Operational model	May detect penetration attempt by individual user who seeks higher privileges.
File access activity		
Read, write, create, delete frequency	Mean and standard deviation	Abnormalities for read and write access for individual users may signify masquerading or browsing.
Records read, written	Mean and standard deviation	Abnormality could signify an attempt to obtain sensitive data by inference and aggregation.
Failure count for read, write, create, delete	Operational	May detect users who persistently attempt to access unauthorized files.

Distributed Intrusion Detection - Architecture



Honeypots

- decoy systems to lure attackers
  - away from accessing critical systems
  - to collect information of their activities
  - to encourage attacker to stay on system so administrator can respond
- are filled with fabricated information

Password Management

- front-line defense against intruders
- users supply both:
  - login – determines privileges of that user
  - password – to identify them
- passwords often stored encrypted
  - Unix uses multiple DES (variant with salt)
  - more recent systems use hash function

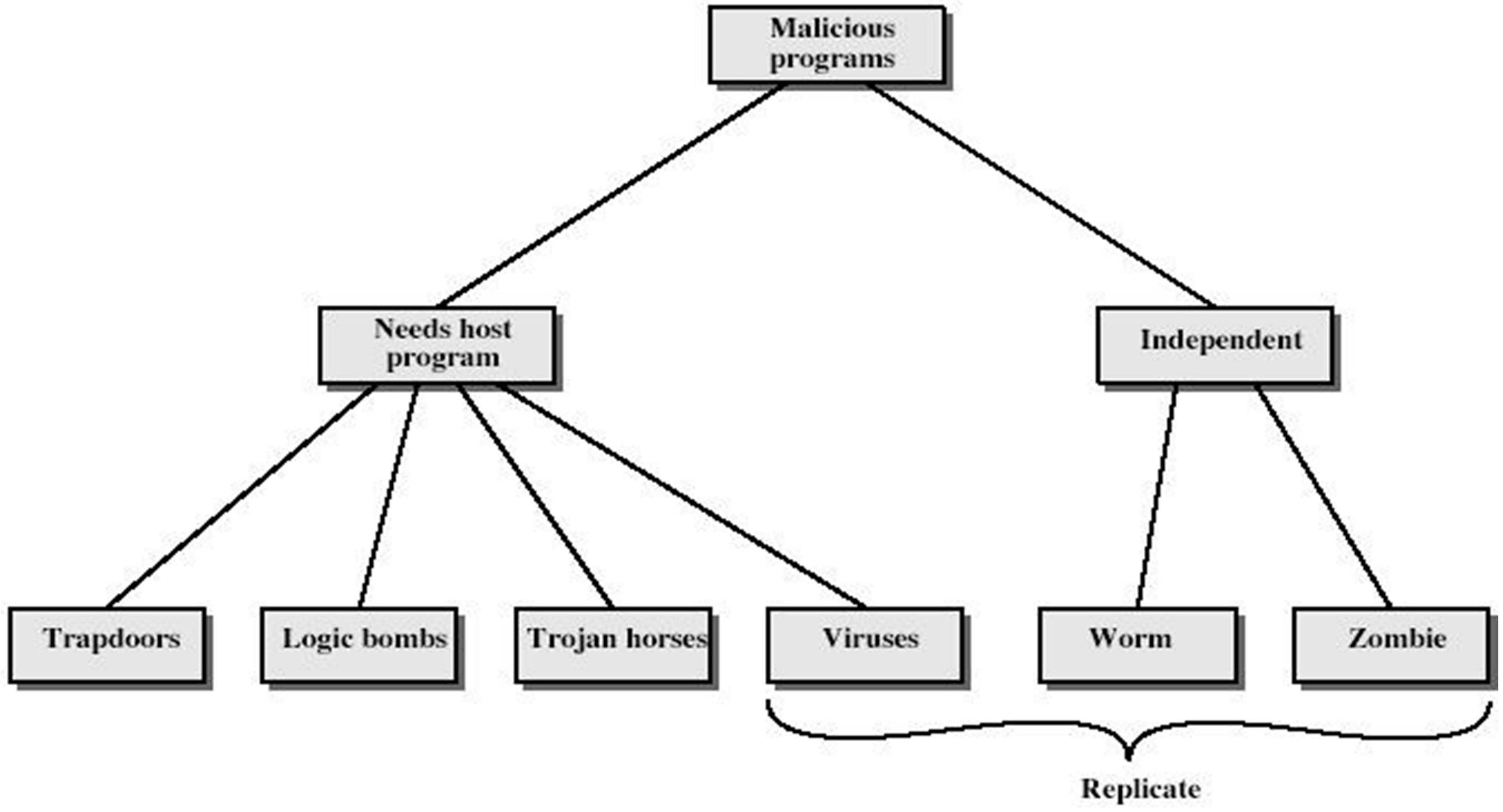
Managing Passwords

- need policies and good user education
- protect password file from general access
- Enforce rules for “good” passwords
- Change password periodically
- Run password –guessing program
- Monitor login failures
- Proactive Password Checking

Viruses and Other Malicious Content

- computer viruses have got a lot of publicity
- one of a family of **malicious software**
- effects usually obvious
- have figured in news reports, fiction, movies (often exaggerated)
- getting more attention than deserve
- are a concern though

Malicious Software



Logic Bomb

- one of oldest types of malicious software
- code embedded in legitimate program
- activated when specified conditions met
  - eg presence/absence of some file
  - particular date/time
  - particular user
- when triggered typically damage system

Trojan Horse

- program with hidden side-effects
- which is usually superficially attractive
  - eg game, s/w upgrade etc
- when run performs some additional tasks
  - allows attacker to indirectly gain access they do not have directly
- often used to propagate a virus/worm or install a backdoor
- or simply to destroy data

Zombie

- program which secretly takes over another networked computer
- then uses it to indirectly launch attacks
- often used to launch distributed denial of service (DDoS) attacks
- exploits known flaws in network systems

Viruses

- a piece of self-replicating code attached to some other code
  - cf biological virus
- both propagates itself & carries a payload
  - carries code to make copies of itself
  - as well as code to perform some covert task

Virus Operation

- virus phases:
  - dormant – waiting on trigger event
  - propagation – replicating to programs/disks
  - triggering – by event to execute payload
  - execution – of payload

Virus Structure

```
program V :=
    {goto main;
    1234567;
    subroutine infect-executable :=   {loop:
            file := get-random-executable-file;
            if (first-line-of-file = 1234567) then goto loop
            else prepend V to file; }
    subroutine do-damage :=      {whatever damage is to be done}
    subroutine trigger-pulled :=  {return true if some condition holds}
    main: main-program :=  {infect-executable;
                    if trigger-pulled then do-damage;
                    goto next;}
    next:
}
```

Macro Virus

- **macro code** attached to some **data file**
- interpreted by program using file
  - eg Word/Excel macros
  - esp. using auto command & command macros
- code is now platform independent
- is a major source of new viral infections

Email Virus

- spread using email with attachment containing a macro virus
- triggered when user opens attachment
- or worse even when mail viewed by using scripting features in mail agent
- usually targeted at Microsoft Outlook mail agent & Word/Excel documents

Worms

- replicating but not infecting program
- typically spreads over a network

Worm Operation

- worm phases like those of viruses:
  - dormant
  - propagation
    - search for other systems to infect
    - establish connection to target remote system
    - replicate self onto remote system
  - triggering
  - execution

Virus Countermeasures

- viral attacks exploit lack of integrity control on systems
- to defend need to add such controls
- typically by one or more of:
  - **prevention** - block virus infection mechanism
  - **detection** - of viruses in infected system
  - **reaction** - restoring system to clean state

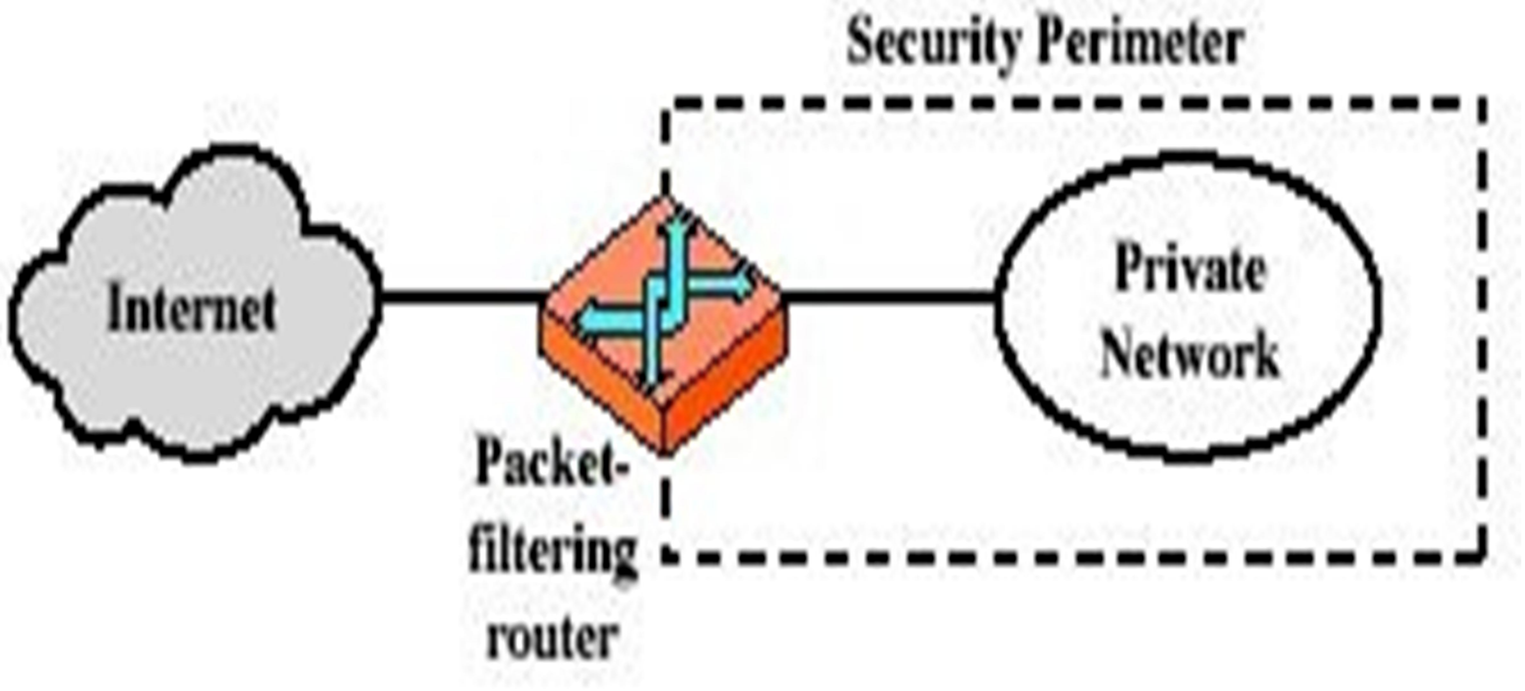
Anti-Virus Software

- **first-generation**
  - scanner uses virus signature to identify virus
  - or change in length of programs
- **second-generation**
  - uses heuristic rules to spot viral infection
  - or uses program checksums to spot changes
- **third-generation**
  - memory-resident programs identify virus by actions
- **fourth-generation**
  - packages with a variety of antivirus techniques
  - eg scanning & activity traps, access-controls

Advanced Anti-Virus Techniques

- generic decryption
  - use CPU simulator to check program signature & behavior before actually running it
- digital immune system (IBM)
  - general purpose emulation & virus detection
  - any virus entering org is captured, analyzed, detection/shielding created for it, removed

Firewalls - Packet Filters



(a) Packet-filtering router

Firewalls – Packet Filters

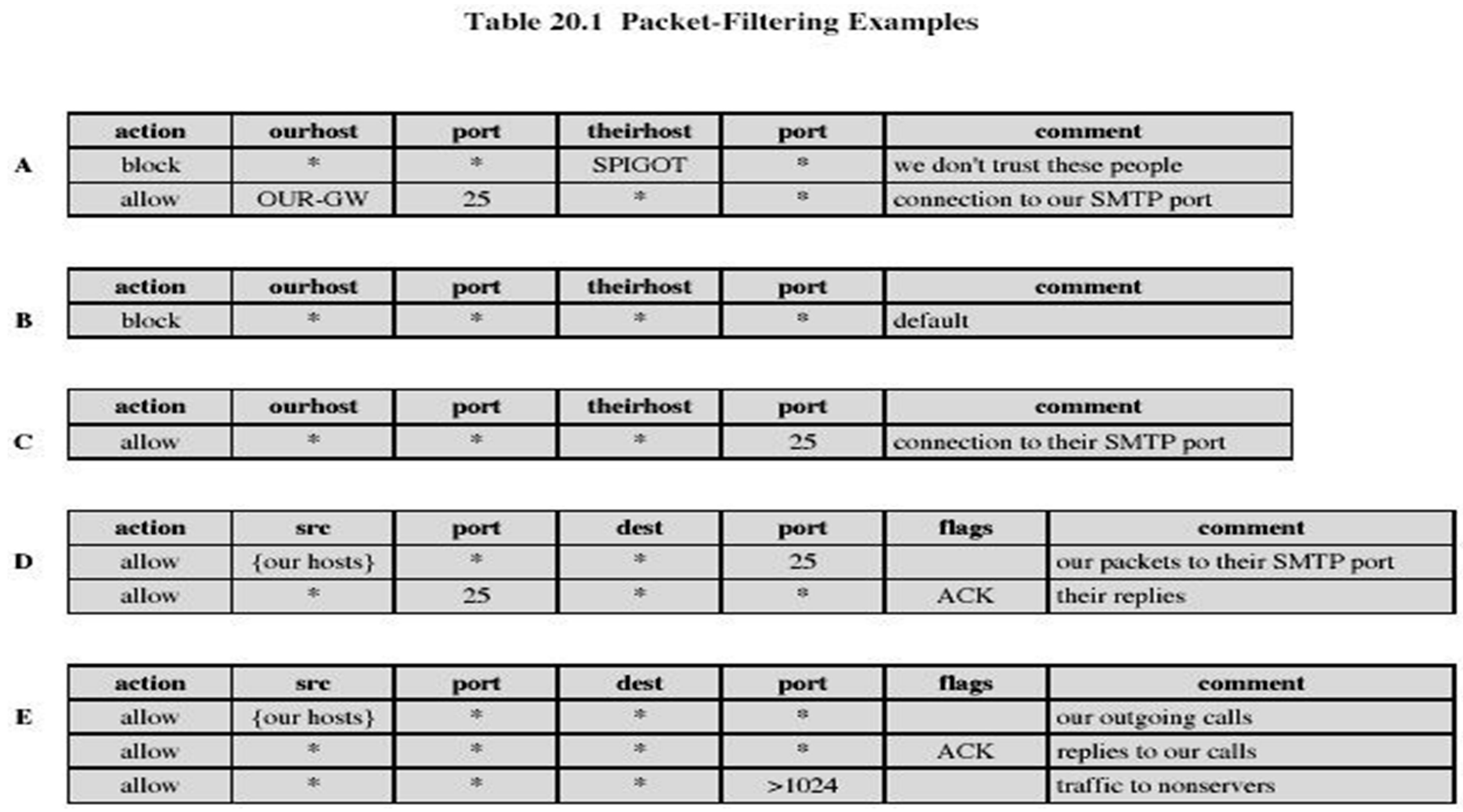


Table 20.1 Packet-Filtering Examples

	action	ourhost	port	theirhost	port	comment
A	block	*	*	SPIGOT	*	we don't trust these people
	allow	OUR-GW	25	*	*	connection to our SMTP port

	action	ourhost	port	theirhost	port	comment
B	block	*	*	*	*	default

	action	ourhost	port	theirhost	port	comment
C	allow	*	*	*	25	connection to their SMTP port

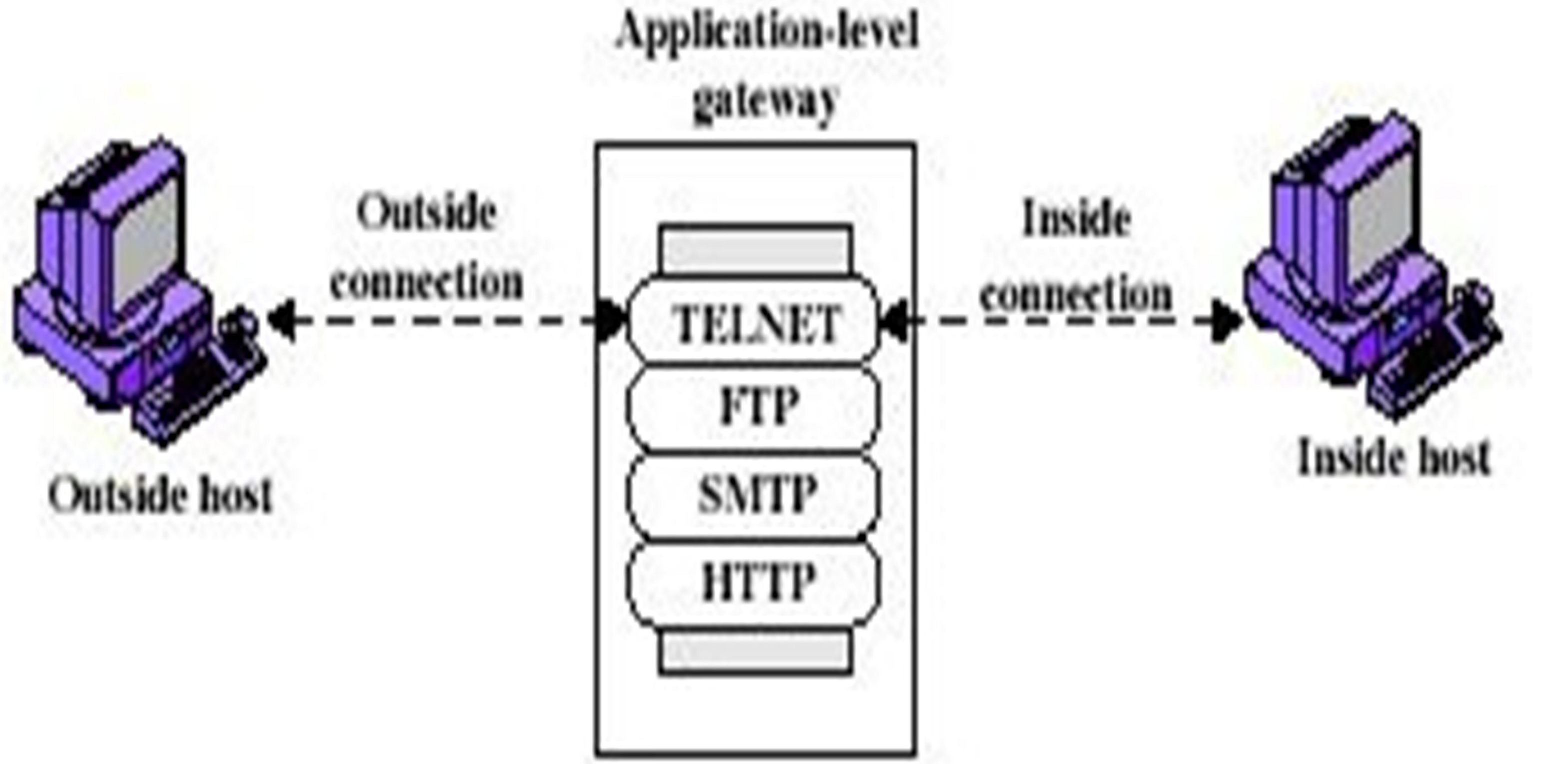
	action	src	port	dest	port	flags	comment
D	allow	{our hosts}	*	*	25		our packets to their SMTP port
	allow	*	25	*	*	ACK	their replies

	action	src	port	dest	port	flags	comment
E	allow	{our hosts}	*	*	*		our outgoing calls
	allow	*	*	*	*	ACK	replies to our calls
	allow	*	*	*	>1024		traffic to nonservers

Attacks on Packet Filters

- IP address spoofing
  - fake source address to be trusted
  - add filters on router to block
- source routing attacks
  - attacker sets a route other than default
  - block source routed packets
- tiny fragment attacks
  - split header info over several tiny packets

Firewalls - Application Level Gateway (or Proxy)

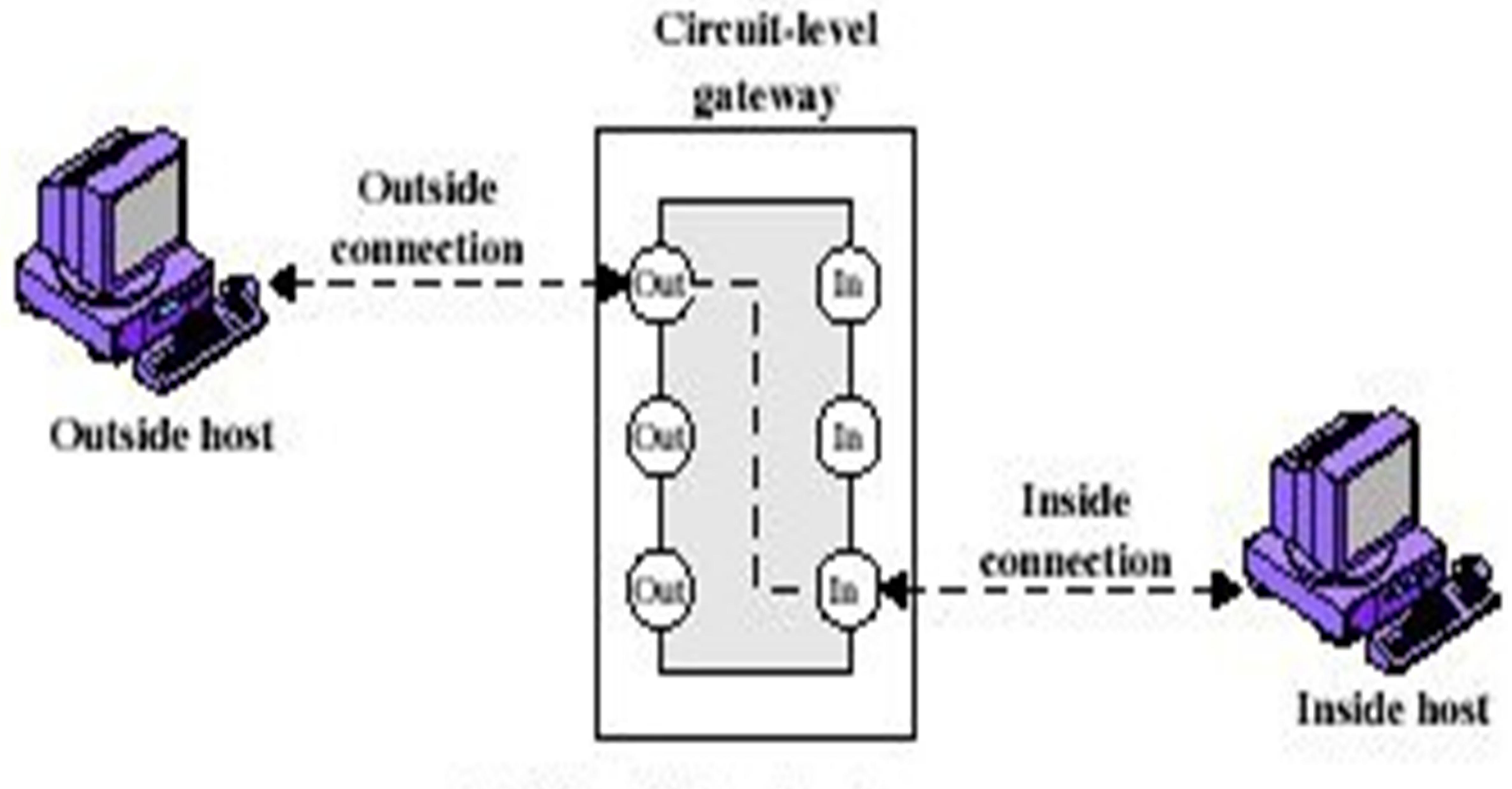


(b) Application-level gateway

Firewalls - Application Level Gateway (or Proxy)

- use an application specific gateway / proxy
- has full access to protocol
  - user requests service from proxy
  - proxy validates request as legal
  - then actions request and returns result to user
- need separate proxies for each service
  - some services naturally support proxying
  - others are more problematic

Firewalls - Circuit Level Gateway



(c) Circuit-level gateway