

Cryptography and Network Security

Unit-2



Modern Block Ciphers

- now look at modern block ciphers
- one of the most widely used types of cryptographic algorithms
- provide secrecy /authentication services
- focus on DES (Data Encryption Standard)
- to illustrate block cipher design principles

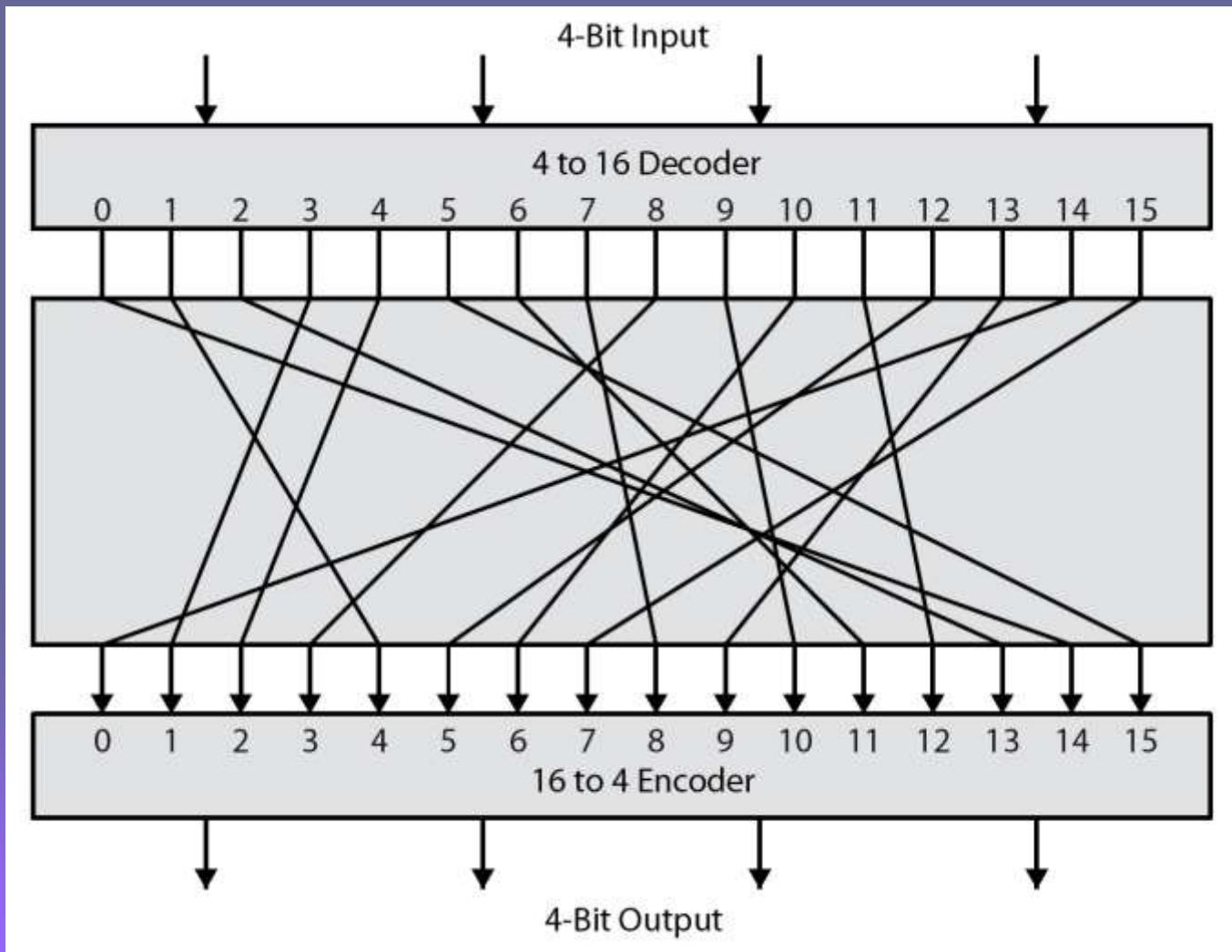
Block vs Stream Ciphers

- block ciphers process messages in 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
- broader range of applications

Block Cipher Principles

- most symmetric block ciphers are based on a **Feistel Cipher Structure**
- needed since must be able to **decrypt** ciphertext to recover messages efficiently
- block ciphers look like an extremely large substitution
- would need table of 2^{64} entries for a 64-bit block
- instead create from smaller building blocks
- using idea of a product cipher

Ideal Block Cipher



Claude Shannon and Substitution-Permutation Ciphers

- Claude Shannon introduced idea of substitution-permutation (S-P) networks in 1949 paper
- form basis of modern block ciphers
- S-P nets are based on the two primitive cryptographic operations seen before:
 - *substitution* (S-box)
 - *permutation* (P-box)
- provide *confusion & diffusion* of message & key

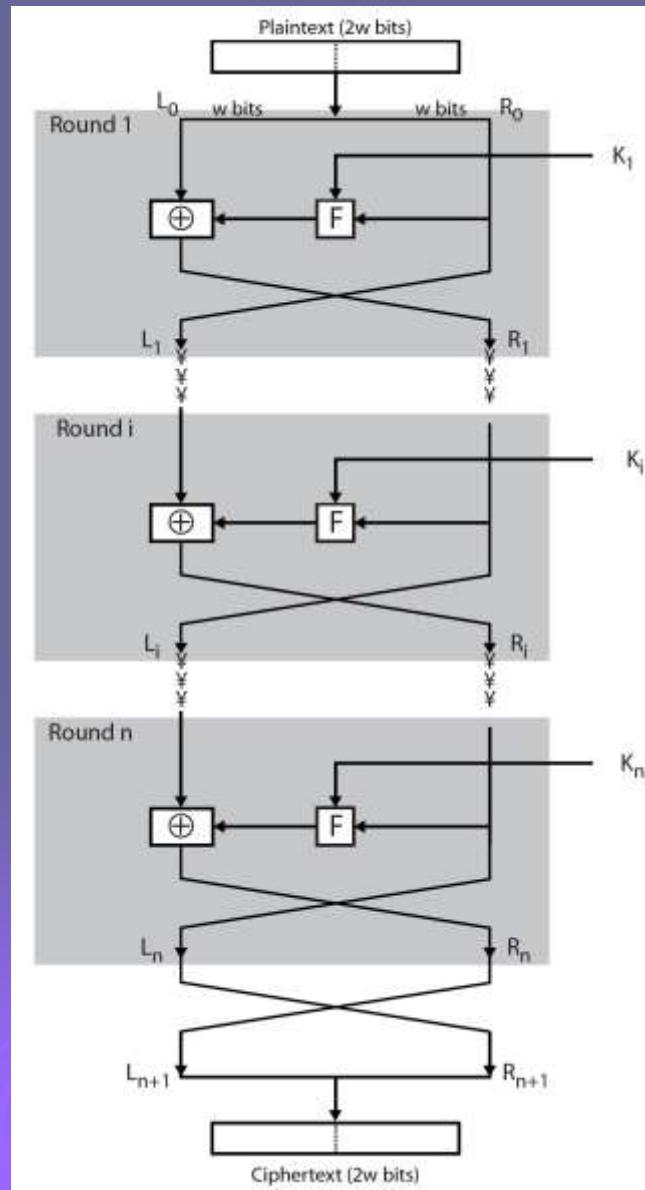
Confusion and Diffusion

- cipher needs to completely obscure statistical properties of original message
- a one-time pad does this
- more practically Shannon suggested combining S & P elements to obtain:
- **diffusion** – dissipates statistical structure of plaintext over bulk of ciphertext
- **confusion** – makes relationship between ciphertext and key as complex as possible

Feistel Cipher Structure

- Horst Feistel devised the **feistel cipher**
 - based on concept of invertible product cipher
- 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
- implements Shannon's S-P net concept

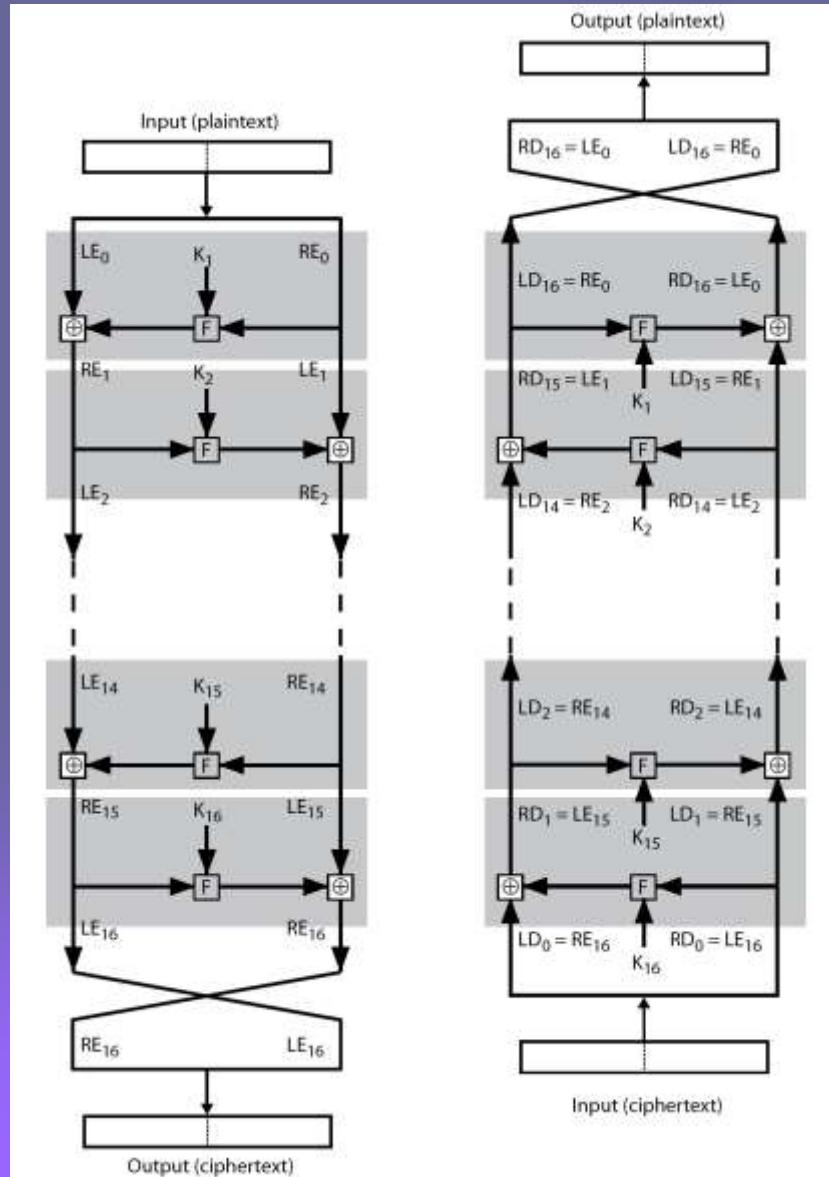
Feistel Cipher Structure



Feistel Cipher Design Elements

- block size
- key size
- number of rounds
- subkey generation algorithm
- round function
- fast software en/decryption
- ease of analysis

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
- has been considerable controversy over its security

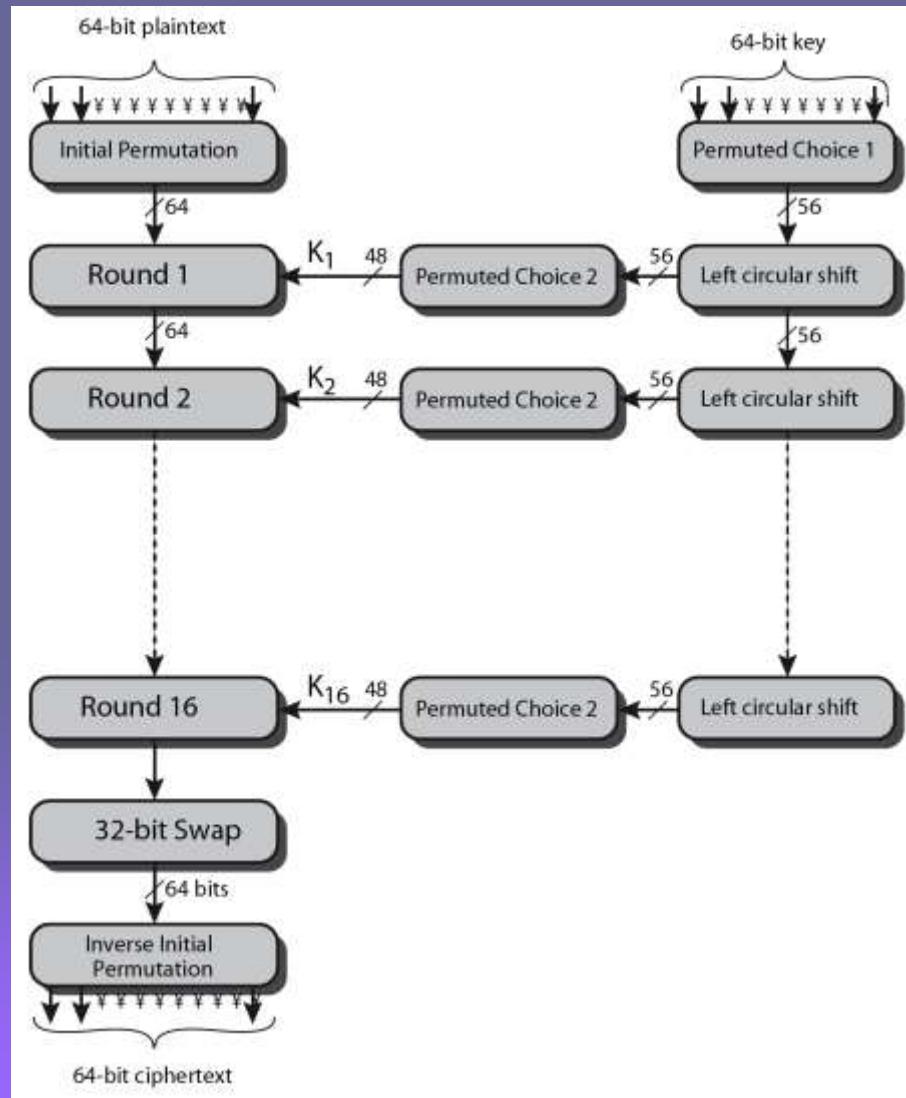
DES History

- IBM developed Lucifer cipher
 - by team led by Feistel in late 60's
 - used 64-bit data blocks with 128-bit key
- then redeveloped as a commercial cipher with input from NSA and others
- in 1973 NBS issued request for proposals for a national cipher standard
- IBM submitted their revised Lucifer which was eventually accepted as the DES

DES Design Controversy

- although DES standard is public
- was considerable controversy over design
 - in choice of 56-bit key (vs Lucifer 128-bit)
 - and because design criteria were classified
- subsequent events and public analysis show in fact design was appropriate
- use of DES has flourished
 - especially in financial applications
 - still standardised for legacy application use

DES Encryption Overview



Initial Permutation IP

- first step of the data computation
- IP reorders the input data bits
- even bits to LH half, odd bits to RH half
- quite regular in structure (easy in h/w)
- example:

IP(675a6967 5e5a6b5a) = (fffb2194d
004df6fb)

DES Round Structure

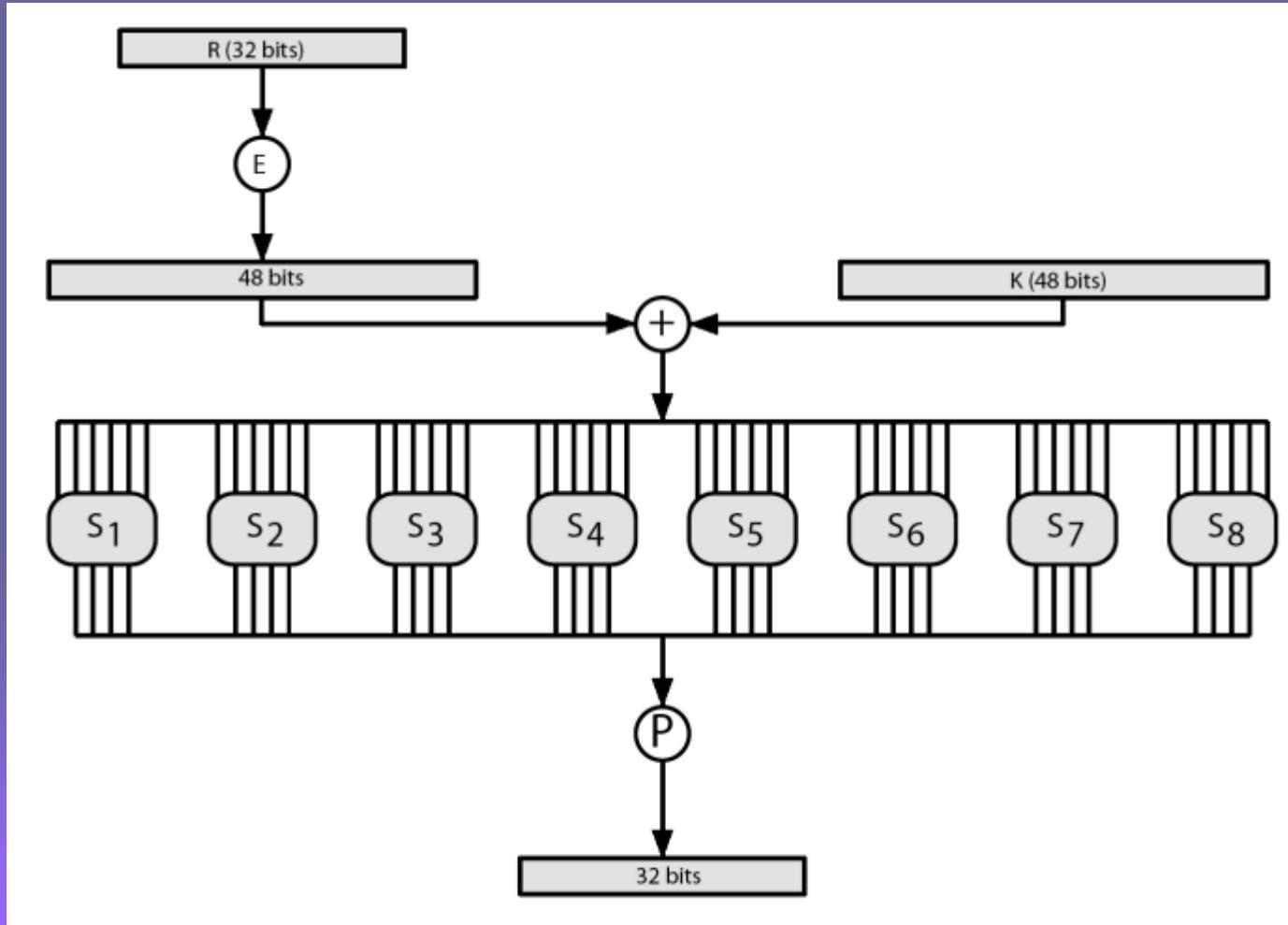
- uses two 32-bit L & R halves
- as for any Feistel cipher can describe as:

$$L_i = R_{i-1}$$

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

- F takes 32-bit R half and 48-bit subkey:
 - expands R to 48-bits using perm E
 - adds to subkey using XOR
 - passes through 8 S-boxes to get 32-bit result
 - finally permutes using 32-bit perm P

DES Round Structure



Substitution Boxes S

- have eight S-boxes which map 6 to 4 bits
- each S-box is actually 4 little 4 bit boxes
 - outer bits 1 & 6 (**row** bits) select one row of 4
 - inner bits 2-5 (**col** bits) are substituted
 - result is 8 lots of 4 bits, or 32 bits
- row selection depends on both data & key
 - feature known as autoclaving (autokeying)
- example:
 - $S(18 \ 09 \ 12 \ 3d \ 11 \ 17 \ 38 \ 39) = 5fd25e03$

DES Key Schedule

- forms subkeys used in each round
 - initial permutation of the key (PC1) which selects 56-bits in two 28-bit halves
 - 16 stages consisting of:
 - rotating **each half** separately either 1 or 2 places depending on the **key rotation schedule K**
 - selecting 24-bits from each half & permuting them by PC2 for use in round function F
- note practical use issues in h/w vs s/w

DES Decryption

- decrypt must unwind steps of data computation
- with Feistel design, do encryption steps again using subkeys in reverse order (SK16 ... SK1)
 - 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

Avalanche Effect

- key desirable property of encryption alg
- where a change of **one** input or key bit results in changing approx **half** output bits
- making attempts to “home-in” by guessing keys impossible
- DES exhibits strong avalanche

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 h/w (EFF) in a few days
 - in 1999 above combined in 22hrs!
- still must be able to recognize plaintext
- must now consider alternatives to DES

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
 - related key attacks

Strength of DES – Timing Attacks

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

Differential Cryptanalysis

- one of the most significant recent (public) advances in cryptanalysis
- known by NSA in 70's cf DES design
- Murphy, Biham & Shamir published in 90's
- powerful method to analyse block ciphers
- used to analyse most current block ciphers with varying degrees of success
- DES reasonably resistant to it, cf Lucifer

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 value of the key
- differential cryptanalysis compares two related pairs of encryptions

Differential Cryptanalysis

Compares Pairs of Encryptions

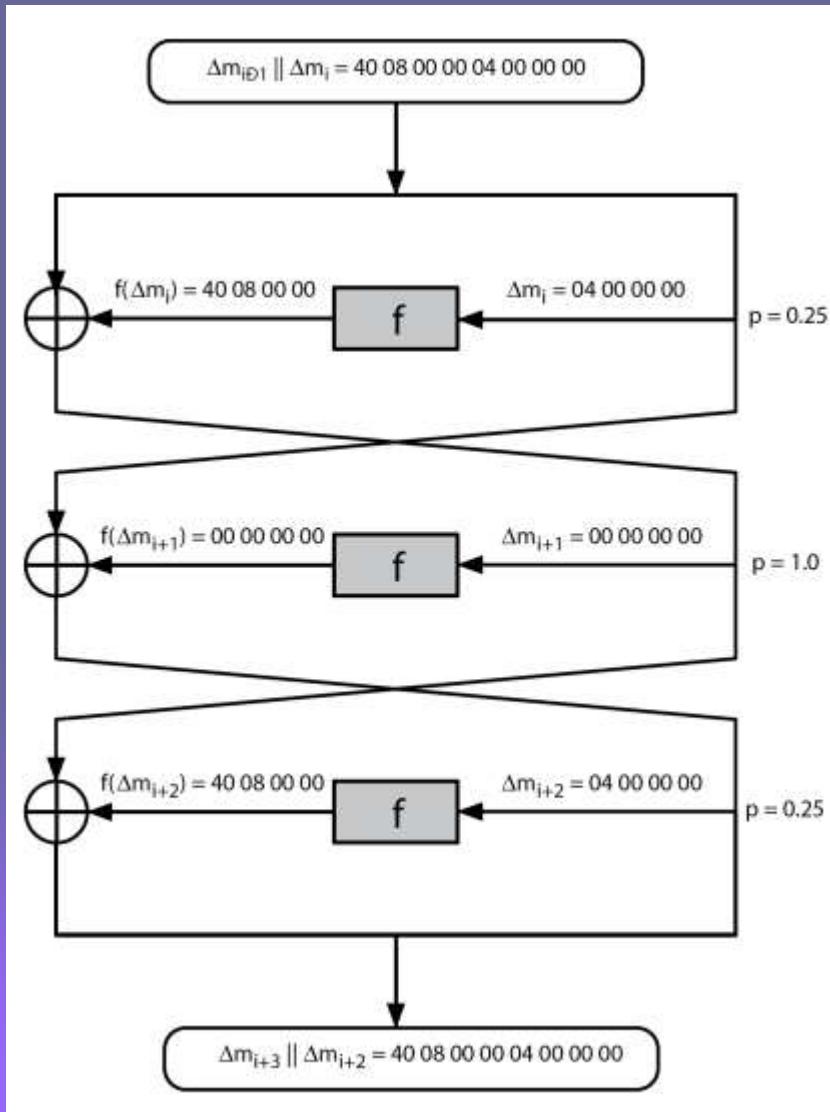
- with a known difference in the input
- searching for a known difference in output
- when same subkeys are used

$$\begin{aligned}\Delta m_{i+1} &= m_{i+1} \oplus m'_{i+1} \\ &= [m_{i-1} \oplus f(m_i, K_i)] \oplus [m'_{i-1} \oplus f(m'_i, K_i)] \\ &= \Delta m_{i-1} \oplus [f(m_i, K_i) \oplus f(m'_i, K_i)]\end{aligned}$$

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 (with decreasing probabilities)

Differential Cryptanalysis



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**, relative ratio is S/N for attack
- can then deduce keys values for the rounds
 - right pairs suggest same key bits
 - wrong pairs give random values
- for large numbers of rounds, probability is so low that more pairs are required than exist with 64-bit inputs
- Biham and Shamir have shown how a 13-round iterated characteristic can break the full 16-round DES

Linear Cryptanalysis

- another recent development
- also a statistical method
- must be iterated over rounds, with decreasing probabilities
- developed by Matsui et al in early 90's
- based on finding linear approximations
- can attack DES with 2^{43} known plaintexts,
easier but still in practise infeasible

Linear Cryptanalysis

- find linear approximations with prob $p \neq \frac{1}{2}$

$$P[i_1, i_2, \dots, i_a] \oplus C[j_1, j_2, \dots, j_b] = K[k_1, k_2, \dots, k_c]$$

where i_a, j_b, k_c are bit locations in P, C, K

- gives linear equation for key bits
- get one key bit using max likelihood alg
- using a large number of trial encryptions
- effectiveness given by: $|p^{-1}/2|$

DES Design Criteria

- as reported by Coppersmith in [COPP94]
- 7 criteria for S-boxes provide for
 - non-linearity
 - resistance to differential cryptanalysis
 - good confusion
- 3 criteria for permutation P provide for
 - increased diffusion

Block Cipher Design

- basic principles still like Feistel's in 1970's
- number of rounds
 - more is better, exhaustive search best attack
- function f:
 - provides "confusion", is nonlinear, avalanche
 - have issues of how S-boxes are selected
- key schedule
 - complex subkey creation, key avalanche

Summary

- have considered:
 - block vs stream ciphers
 - Feistel cipher design & structure
 - DES
 - details
 - strength
 - Differential & Linear Cryptanalysis
 - block cipher design principles

Cryptography and Network Security

Chapter 5

Fourth Edition
by William Stallings

Lecture slides by Lawrie Brown

Chapter 5 –Advanced Encryption Standard

"It seems very simple."

"It is very simple. But if you don't know what the key is it's virtually indecipherable."

— Talking to Strange Men, Ruth Rendell

Origins

- clear a replacement for DES was needed
 - have theoretical attacks that can break it
 - have demonstrated exhaustive key search attacks
- can use Triple-DES – but slow, has small blocks
- US NIST issued call for ciphers in 1997
- 15 candidates accepted in Jun 98
- 5 were shortlisted in Aug-99
- Rijndael was selected as the AES in Oct-2000
- issued as FIPS PUB 197 standard in Nov-2001

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

AES Evaluation Criteria

➤ initial criteria:

- security – effort for practical cryptanalysis
- cost – in terms of computational efficiency
- algorithm & implementation characteristics

➤ final criteria

- general security
- ease of software & hardware implementation
- implementation attacks
- flexibility (in en/decrypt, keying, other factors)

AES Shortlist

- after testing and evaluation, shortlist in Aug-99:
 - MARS (IBM) - complex, fast, high security margin
 - RC6 (USA) - v. simple, v. fast, low security margin
 - Rijndael (Belgium) - clean, fast, good security margin
 - Serpent (Euro) - slow, clean, v. high security margin
 - Twofish (USA) - complex, v. fast, high security margin
- then subject to further analysis & comment
- saw contrast between algorithms with
 - few complex rounds verses many simple rounds
 - which refined existing ciphers verses new proposals

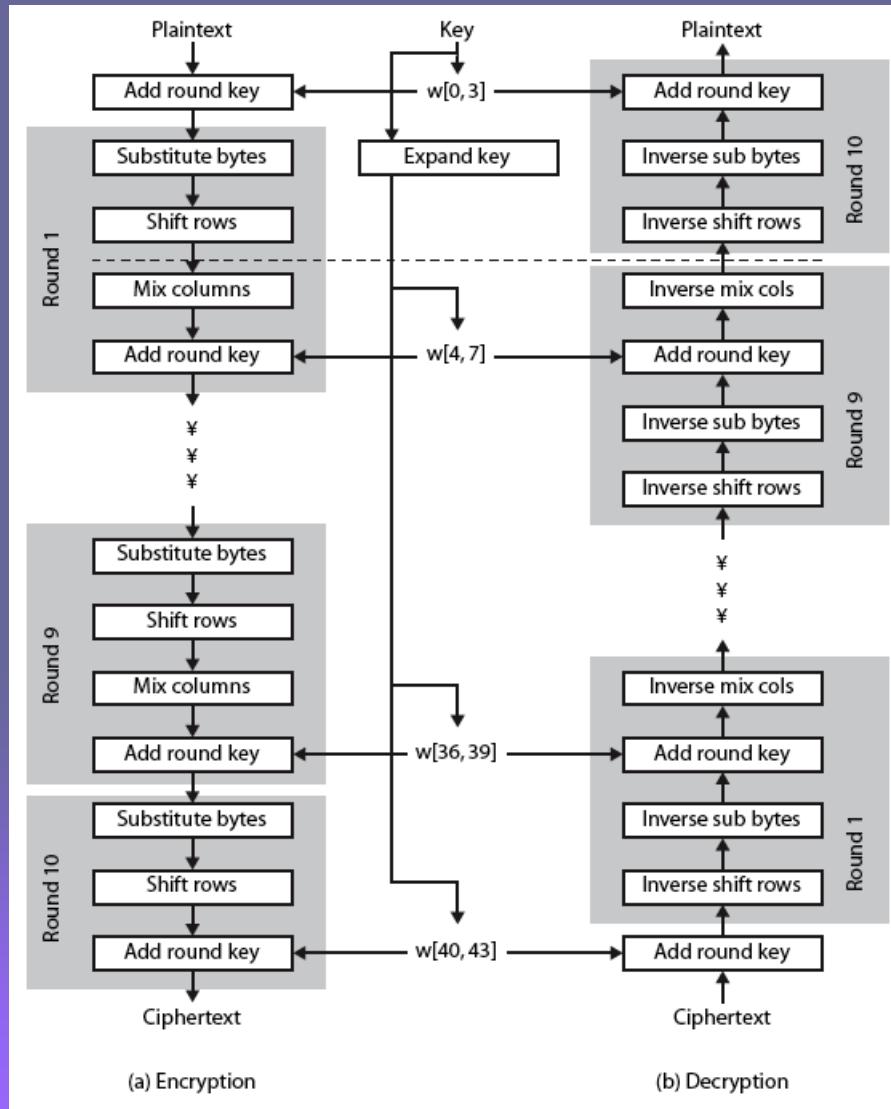
The AES Cipher - Rijndael

- designed by Rijmen-Daemen in Belgium
- has 128/192/256 bit keys, 128 bit data
- an **iterative** rather than **feistel** cipher
 - processes data as block of 4 columns of 4 bytes
 - operates on entire data block in every round
- designed to be:
 - resistant against known attacks
 - speed and code compactness on many CPUs
 - design simplicity

Rijndael

- data block of 4 columns of 4 bytes is state
- key is expanded to array of words
- 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)
 - view as alternating XOR key & scramble data bytes
- initial XOR key material & incomplete last round
- with fast XOR & table lookup implementation

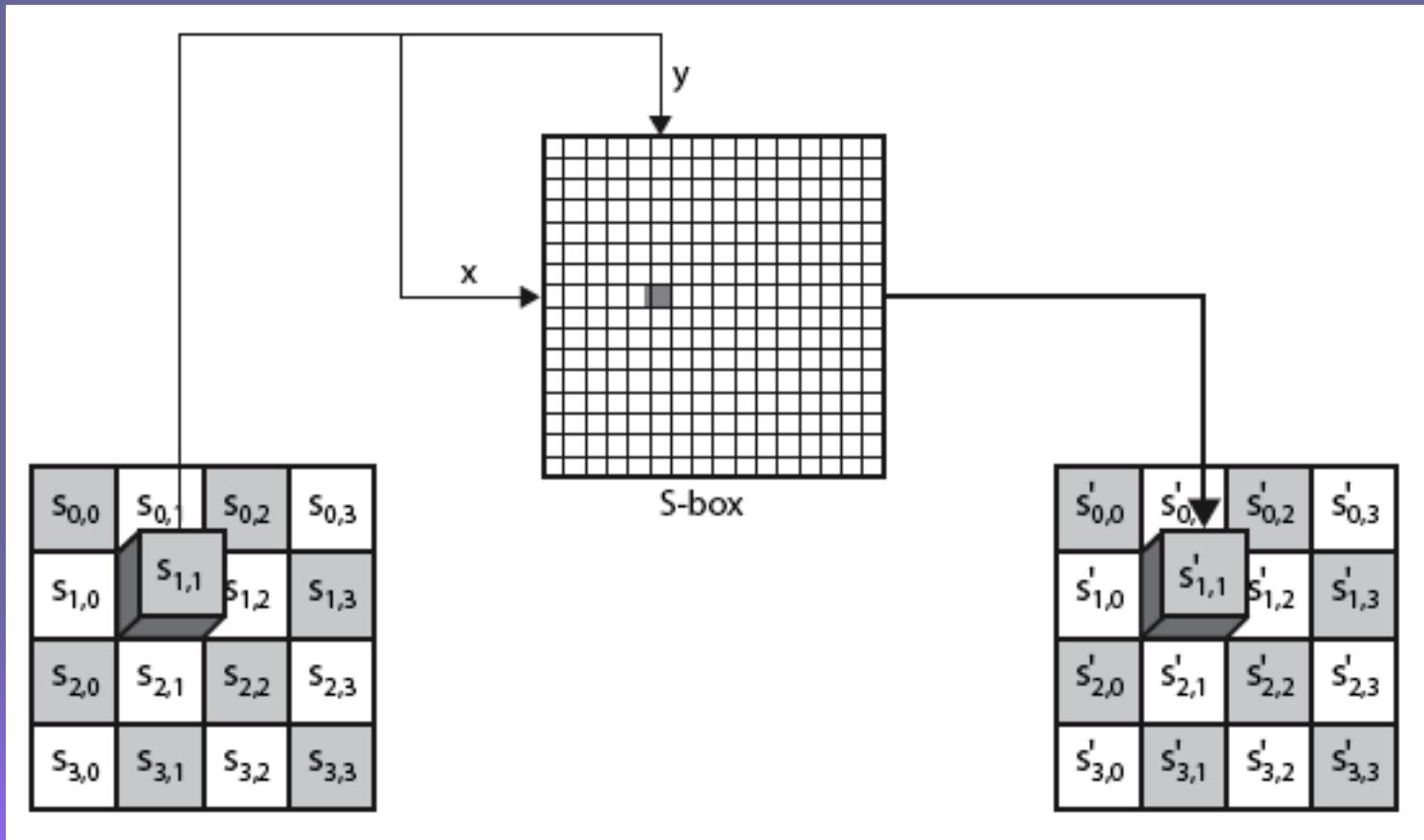
Rijndael



Byte Substitution

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

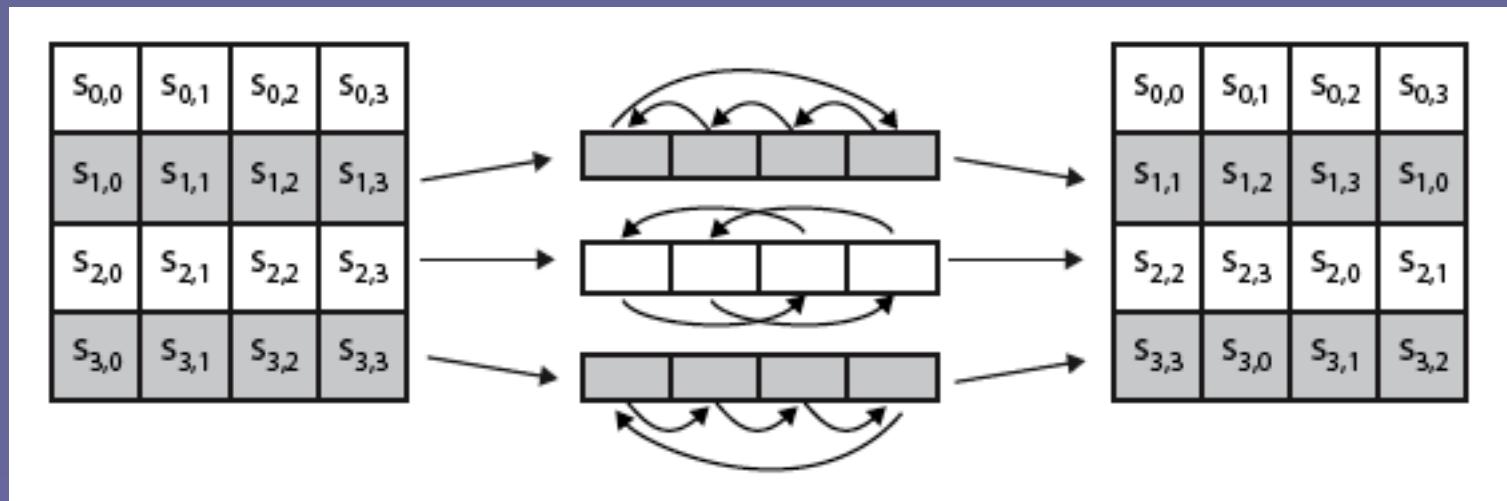
Byte Substitution



Shift Rows

- a circular byte shift in each each
 - 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 inverts using shifts to right
- since state is processed by columns, this step permutes bytes between the columns

Shift Rows

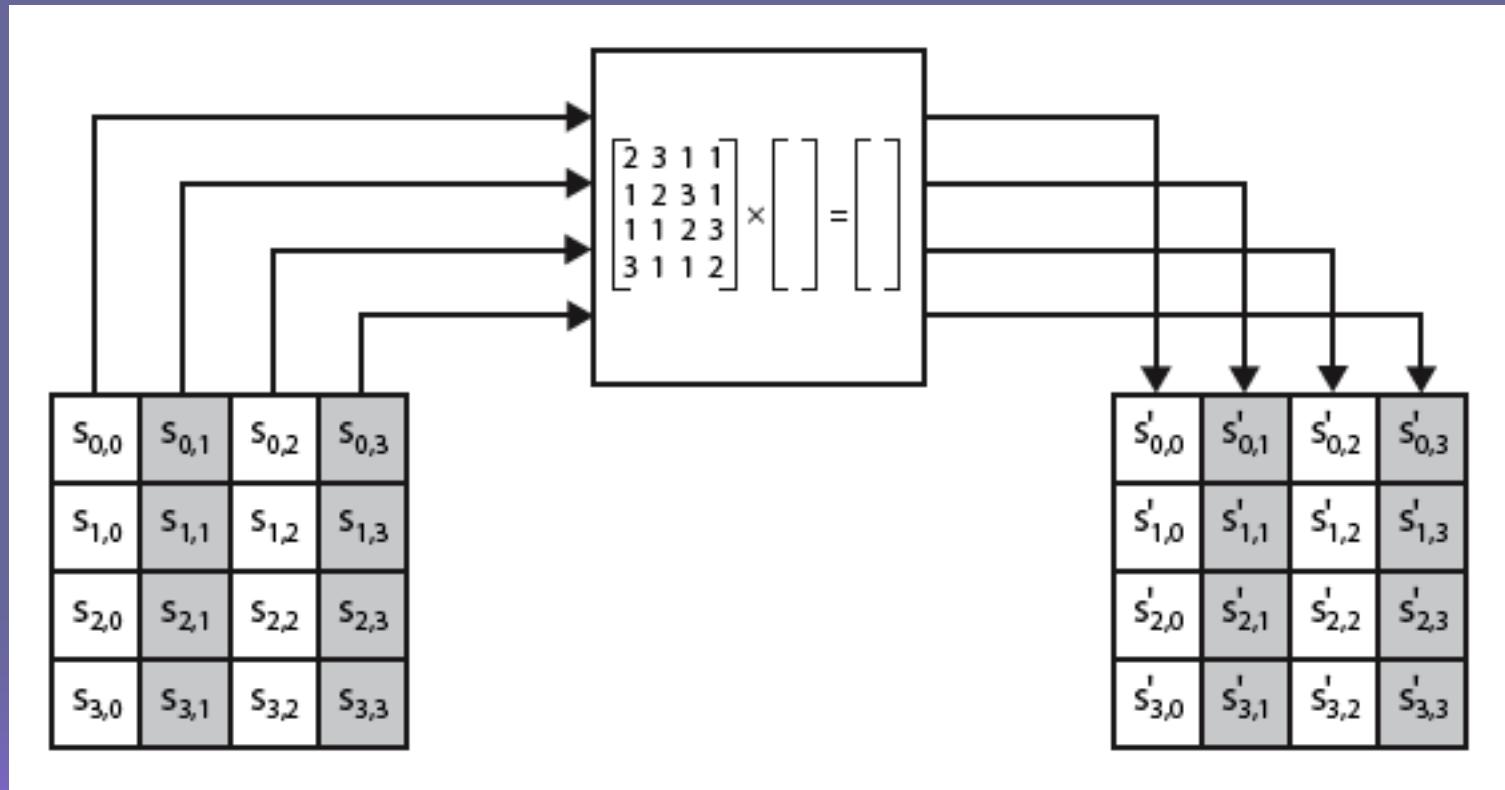


Mix Columns

- 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 $\text{GF}(2^8)$ using prime poly $m(x) = x^8 + x^4 + x^3 + x + 1$

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

Mix Columns



Mix Columns

- can express each col as 4 equations
 - to derive each new byte in col
- decryption requires use of inverse matrix
 - with larger coefficients, hence a little harder
- have an alternate characterisation
 - each column a 4-term polynomial
 - with coefficients in GF(2⁸)
 - and polynomials multiplied modulo (x⁴+1)

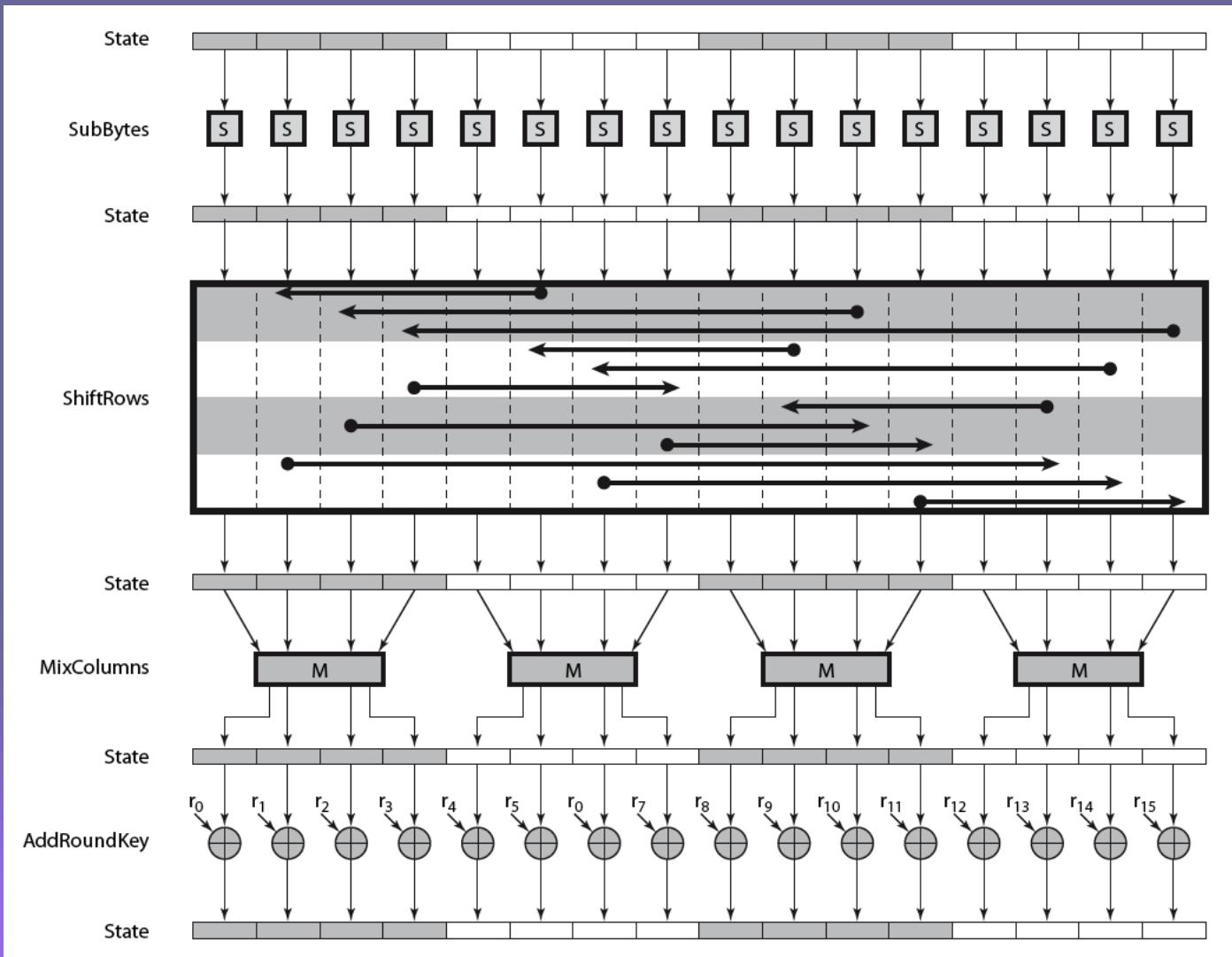
Add Round Key

- XOR state with 128-bits of the round key
- again processed by column (though effectively a series of byte operations)
- inverse for decryption identical
 - since XOR own inverse, with reversed keys
- designed to be as simple as possible
 - a form of Vernam cipher on expanded key
 - requires other stages for complexity / security

Add Round Key

$$\begin{array}{|c|c|c|c|} \hline S_{0,0} & S_{0,1} & S_{0,2} & S_{0,3} \\ \hline S_{1,0} & S_{1,1} & S_{1,2} & S_{1,3} \\ \hline S_{2,0} & S_{2,1} & S_{2,2} & S_{2,3} \\ \hline S_{3,0} & S_{3,1} & S_{3,2} & S_{3,3} \\ \hline \end{array} \oplus \begin{array}{|c|c|c|c|} \hline w_i & w_{i+1} & w_{i+2} & w_{i+3} \\ \hline \end{array} = \begin{array}{|c|c|c|c|} \hline S'_{0,0} & S'_{0,1} & S'_{0,2} & S'_{0,3} \\ \hline S'_{1,0} & S'_{1,1} & S'_{1,2} & S'_{1,3} \\ \hline S'_{2,0} & S'_{2,1} & S'_{2,2} & S'_{2,3} \\ \hline S'_{3,0} & S'_{3,1} & S'_{3,2} & S'_{3,3} \\ \hline \end{array}$$

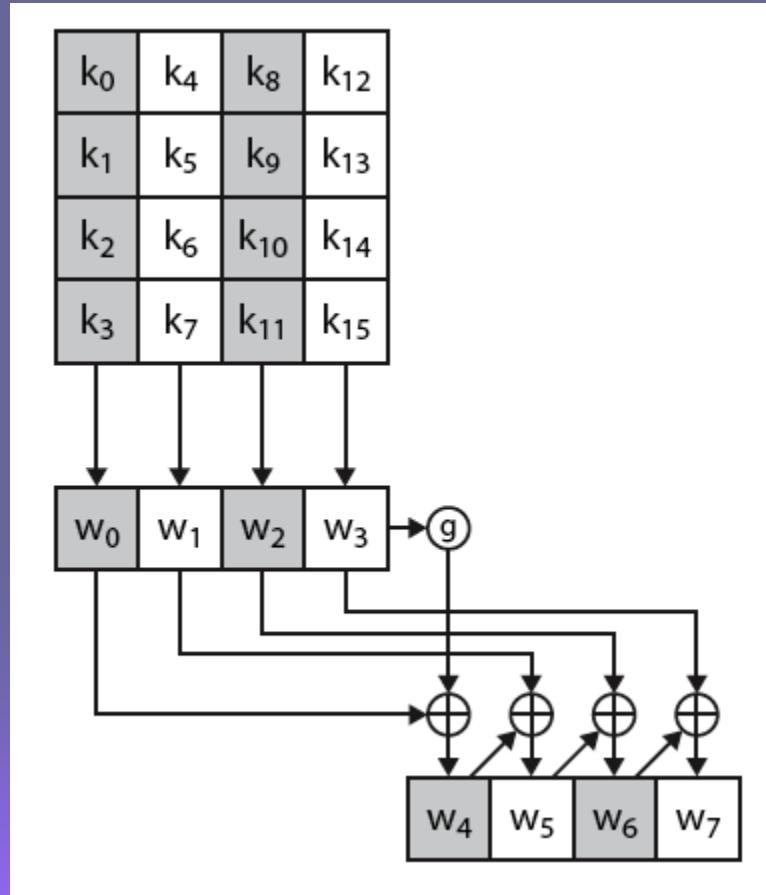
AES Round



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
 - 1st word in 4 has rotate + S-box + XOR round constant on previous, before XOR 4th back

AES Key Expansion



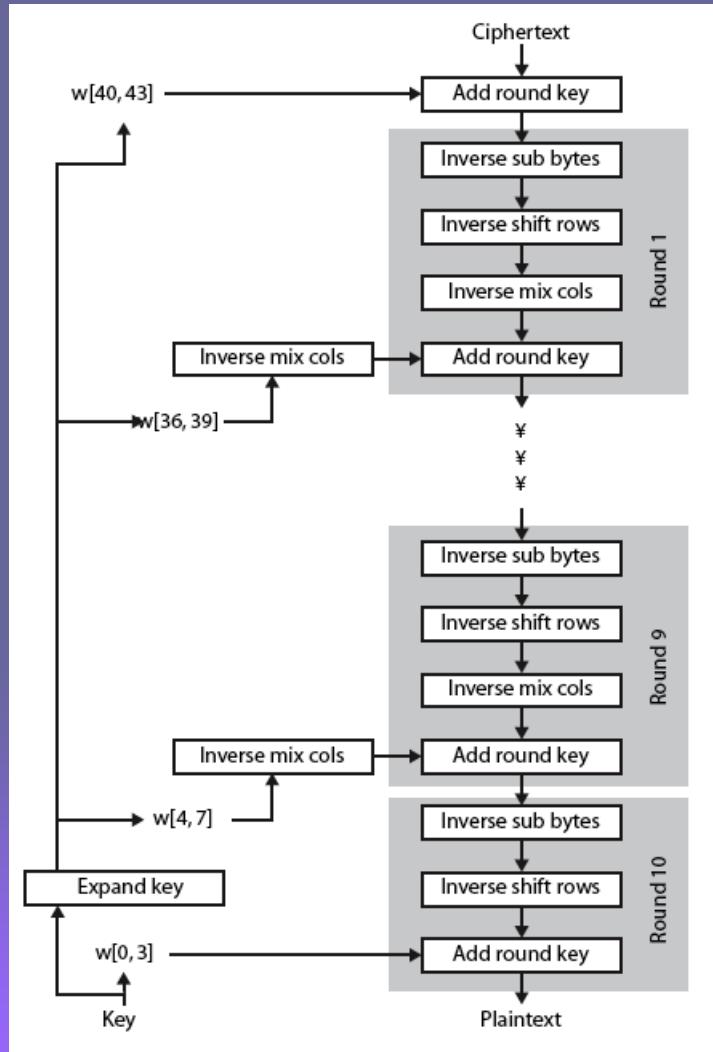
Key Expansion Rationale

- designed to resist known attacks
- design criteria included
 - knowing part key insufficient to find many more
 - invertible transformation
 - fast on wide range of CPU's
 - use round constants to break symmetry
 - diffuse key bits into round keys
 - enough non-linearity to hinder analysis
 - simplicity of description

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

AES Decryption



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 shift
 - add round key works on byte XOR's
 - mix columns requires matrix multiply in $\text{GF}(2^8)$ which works on byte values, can be simplified to use table lookups & byte XOR's

Implementation Aspects

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

Summary

- have considered:
 - the AES selection process
 - the details of Rijndael – the AES cipher
 - looked at the steps in each round
 - the key expansion
 - implementation aspects

Cryptography and Network Security

Chapter 6

Fourth Edition
by William Stallings

Lecture slides by Lawrie Brown

Chapter 6 – Contemporary Symmetric Ciphers

"I am fairly familiar with all the forms of secret writings, and am myself the author of a trifling monograph upon the subject, in which I analyze one hundred and sixty separate ciphers," said Holmes.

— *The Adventure of the Dancing Men,*
Sir Arthur Conan Doyle

Multiple Encryption & DES

- clear a replacement for DES was needed
 - theoretical attacks that can break it
 - demonstrated exhaustive key search attacks
- AES is a new cipher alternative
- prior to this alternative was to use multiple encryption with DES implementations
- Triple-DES is the chosen form

Double-DES?

- could use 2 DES encrypts on each block
 - $C = E_{K_2}(E_{K_1}(P))$
- issue of reduction to single stage
- and have “meet-in-the-middle” attack
 - works whenever use a cipher twice
 - since $X = E_{K_1}(P) = D_{K_2}(C)$
 - 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
- but can use 2 keys with E-D-E sequence
 - $C = E_{K1}(D_{K2}(E_{K1}(P)))$
 - nb encrypt & decrypt equivalent in security
 - if $K1=K2$ then can work with single DES
- standardized in ANSI X9.17 & ISO8732
- no current known practical attacks

Triple-DES with Three-Keys

- although there are no practical attacks on two-key Triple-DES, there have been 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, eg PGP, S/MIME

Modes of Operation

- block ciphers encrypt fixed size blocks
 - eg. DES encrypts 64-bit blocks with 56-bit key
- need some way to en/decrypt arbitrary amounts of data in practise
- **ANSI X3.106-1983 Modes of Use** (now FIPS 81) defines 4 possible modes
- subsequently 5 defined for AES & DES
- have **block** and **stream** modes

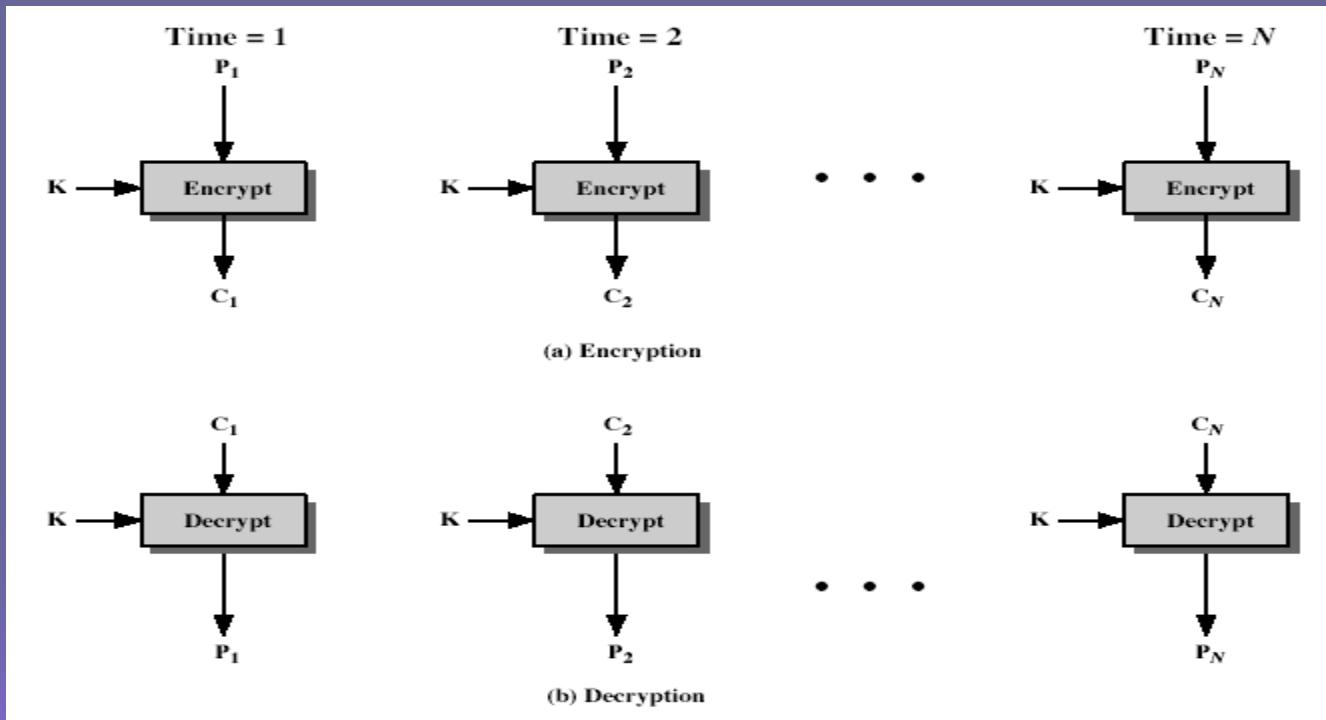
Electronic Codebook Book (ECB)

- message is broken into independent blocks which are encrypted
- each block is a value which is substituted, like a codebook, hence name
- each block is encoded independently of the other blocks

$$C_i = \text{DES}_{K1}(P_i)$$

- uses: secure transmission of single values

Electronic Codebook Book (ECB)



Advantages and Limitations of ECB

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

Cipher Block Chaining (CBC)

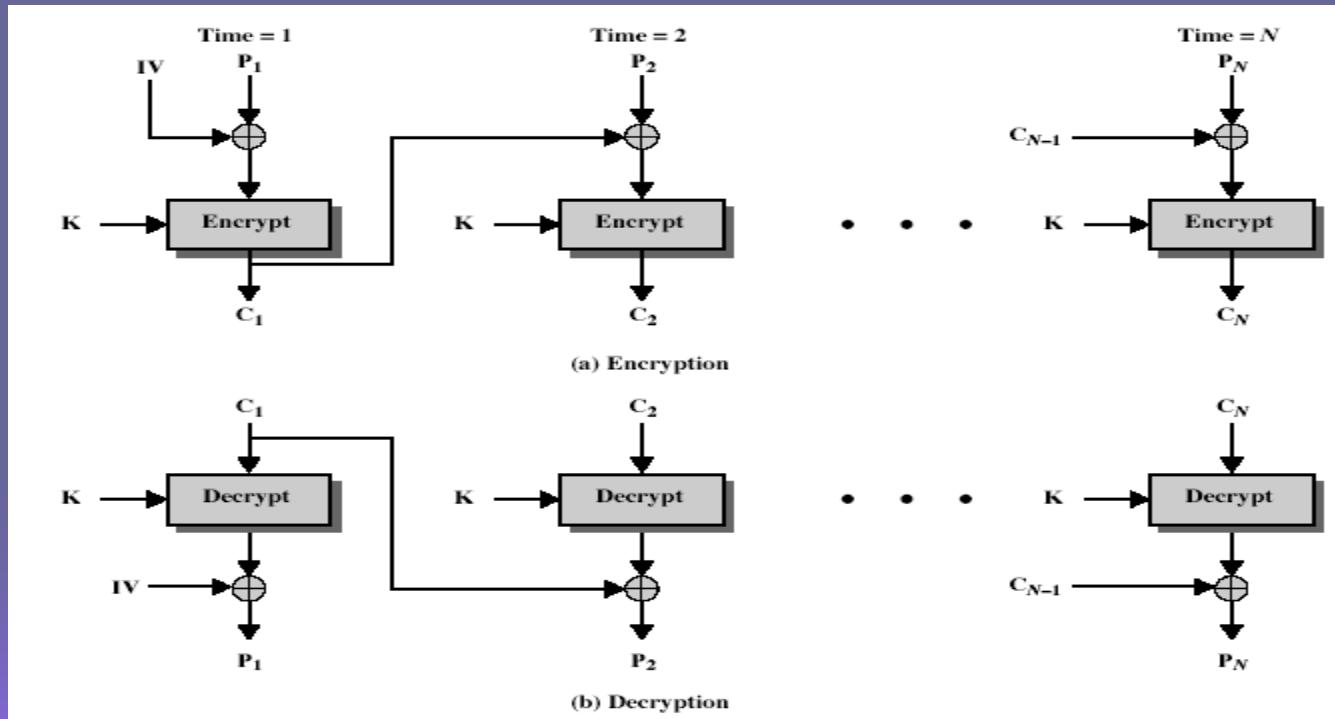
- message is broken into blocks
- linked together in encryption operation
- each previous cipher blocks is chained with current plaintext block, hence name
- use Initial Vector (IV) to start process

$$C_i = \text{DES}_{K1}(P_i \text{ XOR } C_{i-1})$$

$$C_{-1} = \text{IV}$$

- uses: bulk data encryption, authentication

Cipher Block Chaining (CBC)



Message Padding

- at end of message must handle a possible last short block
 - which is not as large as blocksize of cipher
 - pad either with known non-data value (eg nulls)
 - or pad last block along with count of pad size
 - eg. [b1 b2 b3 0 0 0 5]
 - means have 3 data bytes, then 5 bytes pad+count
 - this may require an extra entire block over those in message
- there are other, more esoteric modes, which avoid the need for an extra block

Advantages and Limitations of CBC

- a ciphertext block depends on **all** blocks before it
- any change to a block affects all following ciphertext blocks
- need **Initialization Vector (IV)**
 - which must be known to sender & receiver
 - if sent in clear, attacker can change bits of first block, and change IV to compensate
 - hence IV must either be a fixed value (as in EFTPOS)
 - or must be sent encrypted in ECB mode before rest of message

Cipher FeedBack (CFB)

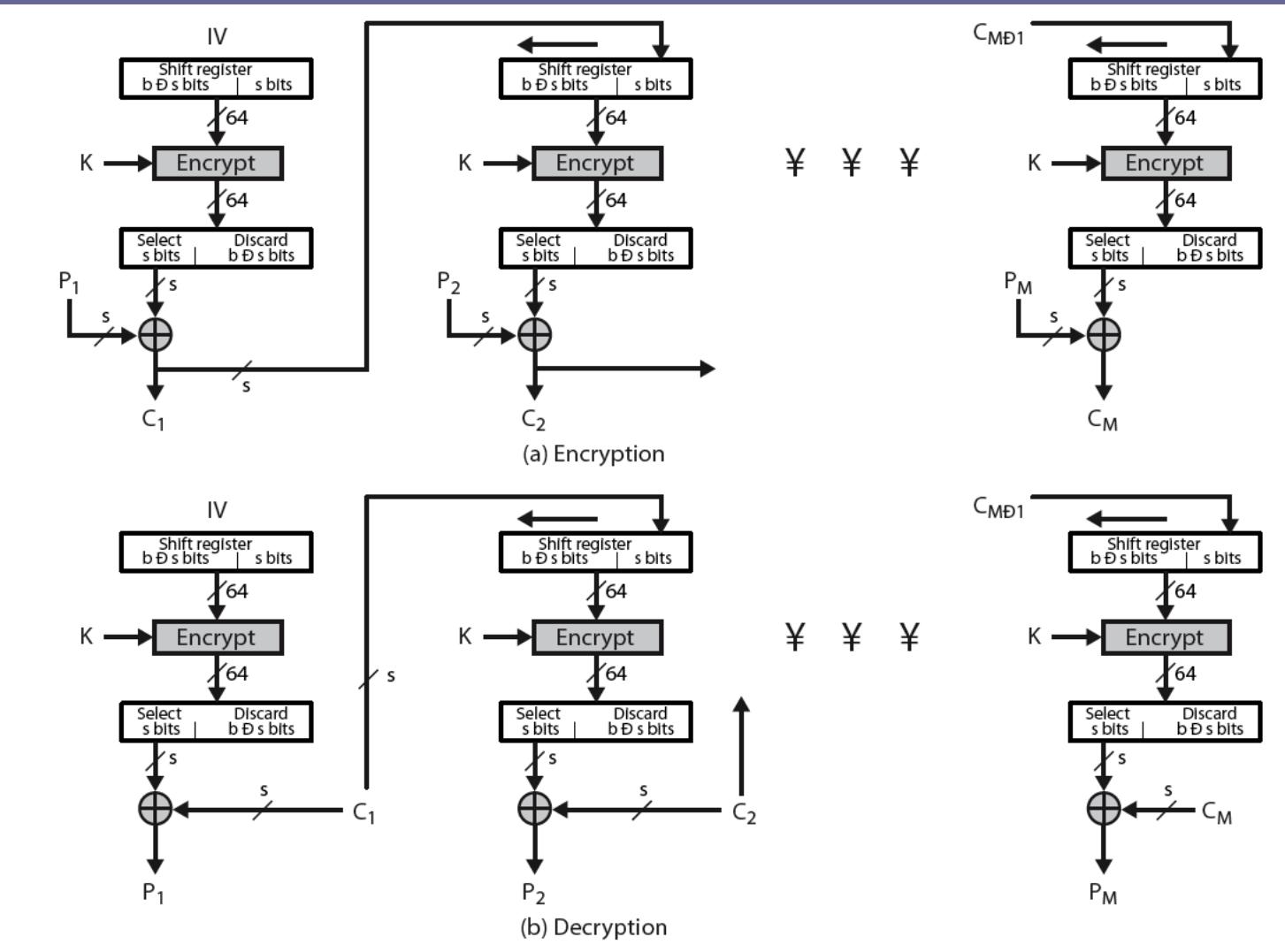
- message is treated as a stream of bits
- added to the output of the block cipher
- result is feed back for next stage (hence name)
- standard allows any number of bit (1,8, 64 or 128 etc) to be feed back
 - denoted CFB-1, CFB-8, CFB-64, CFB-128 etc
- most efficient to use all bits in block (64 or 128)

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

$$C_{-1} = IV$$

- uses: stream data encryption, authentication

Cipher FeedBack (CFB)



Advantages and Limitations of CFB

- appropriate when data arrives in bits/bytes
- most common stream mode
- limitation is need to stall while do block encryption after every n-bits
- note that the block cipher is used in **encryption** mode at **both** ends
- errors propagate for several blocks after the error

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

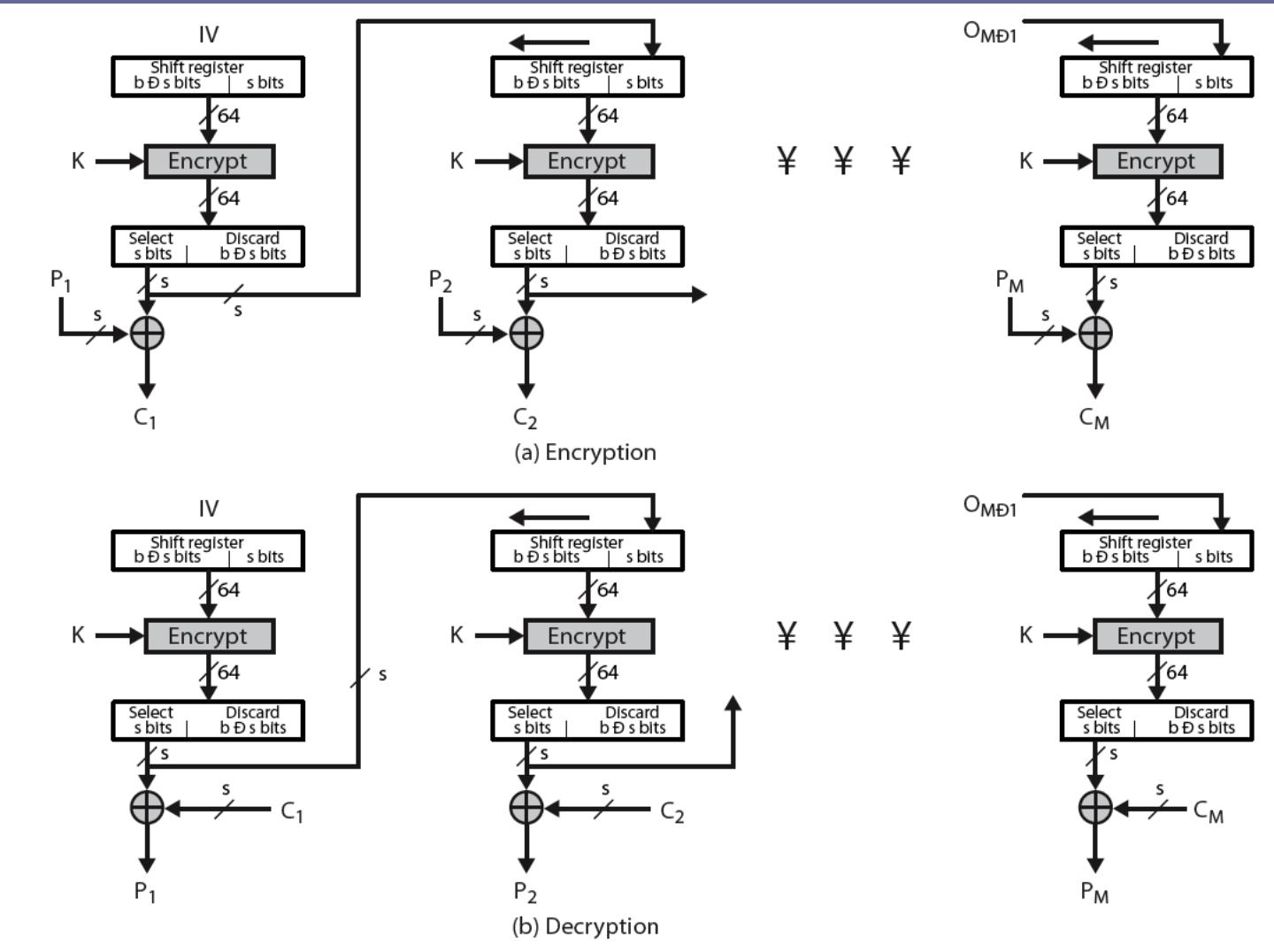
$$C_i = P_i \text{ XOR } O_i$$

$$O_i = \text{DES}_{K1}(O_{i-1})$$

$$O_{-1} = \text{IV}$$

- uses: stream encryption on noisy channels

Output FeedBack (OFB)



Advantages and Limitations of OFB

- bit errors do not propagate
- more vulnerable to message stream modification
- a variation of a Vernam cipher
 - hence must **never** reuse the same sequence (key+IV)
- sender & receiver must remain in sync
- originally specified with m-bit feedback
- subsequent research has shown that only **full block feedback** (ie CFB-64 or CFB-128) should ever be used

Counter (CTR)

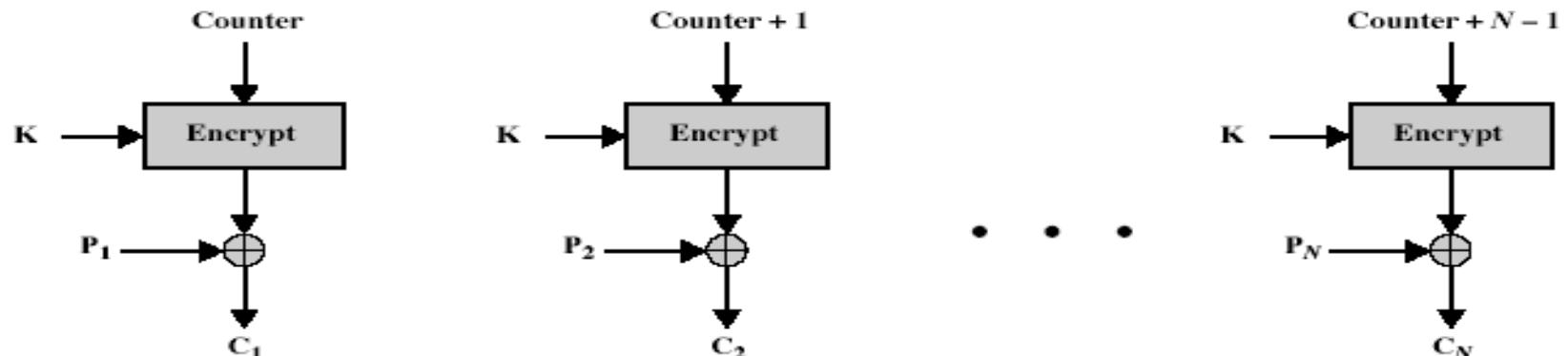
- a “new” mode, though proposed early on
- similar to OFB but encrypts counter value rather than any feedback value
- must have a different key & counter value for every plaintext block (never reused)

$$C_i = P_i \text{ XOR } O_i$$

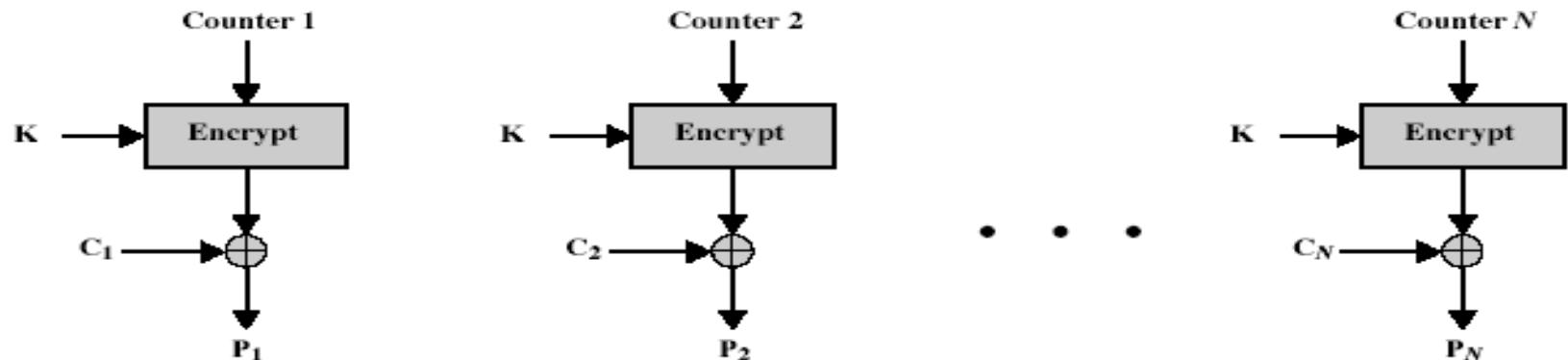
$$O_i = \text{DES}_{K1}(i)$$

- uses: high-speed network encryptions

Counter (CTR)



(a) Encryption



(b) Decryption

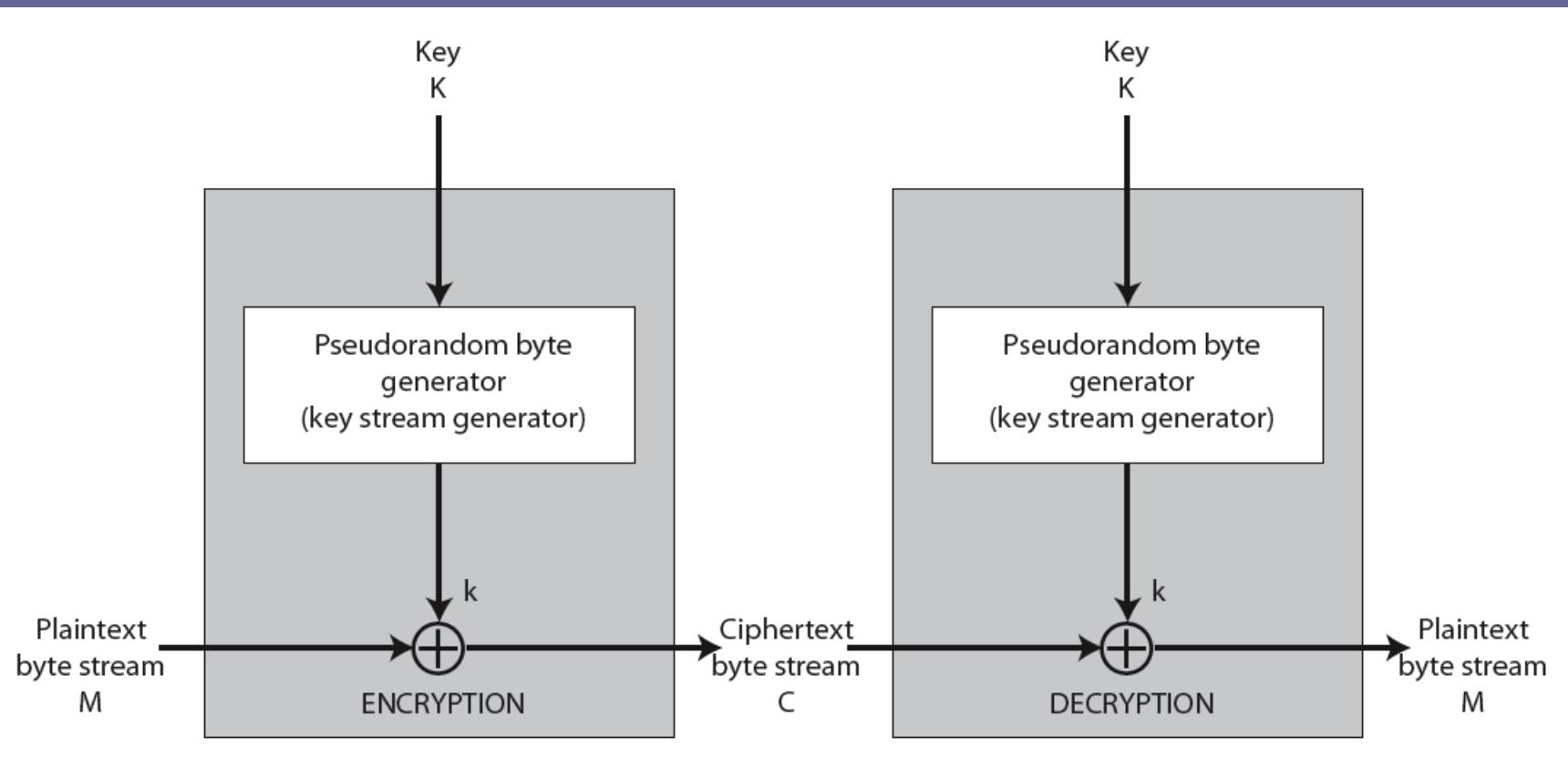
Advantages and Limitations of CTR

- efficiency
 - can do parallel encryptions in h/w or s/w
 - can preprocess in advance of need
 - good for bursty high speed links
- random access to encrypted data blocks
- provable security (good as other modes)
- but must ensure never reuse key/counter values, otherwise could break (cf OFB)

Stream Ciphers

- process message bit by bit (as a stream)
- have a pseudo random **keystream**
- combined (XOR) with plaintext bit by bit
- randomness of **stream key** completely destroys statistically properties in message
 - $C_i = M_i \text{ XOR StreamKey}_i$
- but must never reuse stream key
 - otherwise can recover messages (cf book cipher)

Stream Cipher Structure



Stream Cipher Properties

- some design considerations are:
 - long period with no repetitions
 - statistically random
 - depends on large enough key
 - large linear complexity
- properly designed, can be as secure as a block cipher with same size key
- but usually simpler & faster

RC4

- a proprietary cipher owned by RSA DSI
- another Ron Rivest design, simple but effective
- variable key size, byte-oriented stream cipher
- widely used (web SSL/TLS, wireless WEP)
- key forms random permutation of all 8-bit values
- uses that permutation to scramble input info processed a byte at a time

RC4 Key Schedule

- starts with an array S of numbers: 0..255
- use key to well and truly shuffle
- S forms **internal state** of the cipher

```
for i = 0 to 255 do
    S[i] = i
    T[i] = K[i mod keylen])
j = 0
for i = 0 to 255 do
    j = (j + S[i] + T[i]) (mod 256)
    swap (S[i], S[j])
```

RC4 Encryption

- encryption continues shuffling array values
- sum of shuffled pair selects "stream key" value from permutation
- XOR S[t] with next byte of message to en/decrypt

i = j = 0

for each message byte M_i

i = (i + 1) $\pmod{256}$

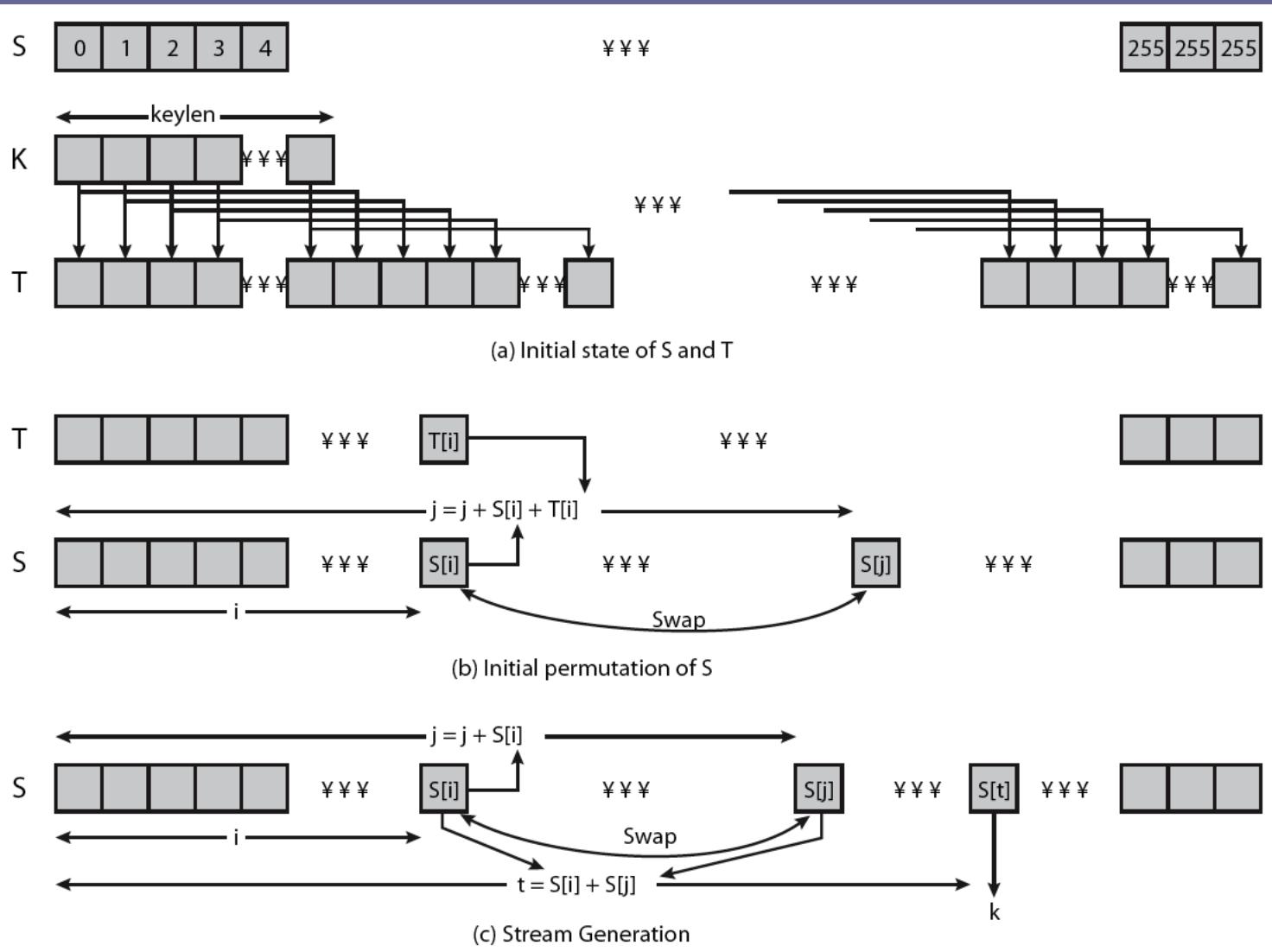
j = (j + S[i]) $\pmod{256}$

swap(S[i], S[j])

t = (S[i] + S[j]) $\pmod{256}$

$C_i = M_i \text{ XOR } S[t]$

RC4 Overview

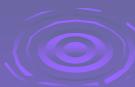


RC4 Security

- 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**
- have a concern with WEP, but due to key handling rather than RC4 itself

Summary

- Triple-DES
- Modes of Operation
 - ECB, CBC, CFB, OFB, CTR
- stream ciphers
- RC4



UNIT II SYMMETRIC KEY CRYPTOGRAPHY

MATHEMATICS OF SYMMETRIC KEY CRYPTOGRAPHY: Algebraic Structures - Modular arithmetic - Euclid's Algorithm - Congruence and Matrices - Groups, Rings, Fields - Finite fields - **SYMMETRIC KEY CIPHERS:** SDES - Block cipher principles of DES - Strength of DES - Differential and Linear Cryptanalysis - Block cipher design principles - Block cipher mode of operation - Evaluation criteria of AES - Advanced Encryption Standard - RC4 - Key Distribution

2.1 ALGEBRAIC STRUCTURES

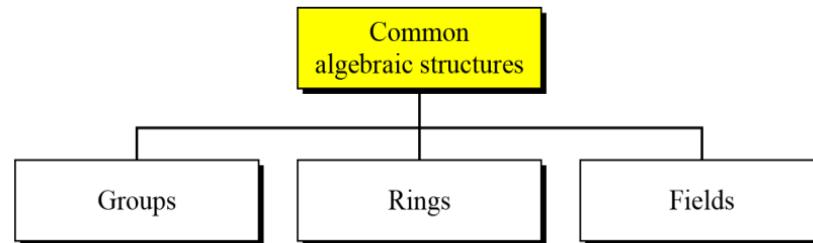


Figure 2.1 Common Algebraic Structures

2.1.1 Groups, Rings, Fields

Groups, rings, and fields are the fundamental elements of a branch of mathematics known as abstract algebra, or modern algebra.

Groups

A group G , sometimes denoted by $\{G, *\}$, is a set of elements with a binary operation denoted by $*$ that associates to each ordered pair (a,b) of elements G in an element $(a*b)$ in G , such that the following axioms are obeyed:

- (A1) **Closure:** If a and b belong to G , then $a*b$ is also in G .
- (A2) **Associative:** $a*(b*c)=(a*b)*c$ for all a, b, c in G .
- (A3) **Identity element:** There is an element e in G such that $a*e=e*a=a$ for all a in G .
- (A4) **Inverse element:** For each a in G , there is an element a' in G such that $a*a'=a'*a=e$.

If a group has a finite number of elements, it is referred to as a **finite group**, and the **order** of the group is equal to the number of elements in the group. Otherwise, the group is an **infinite group**.

A group is said to be **abelian** if it satisfies the following additional condition:

- (A5) **Commutative:** $a*b = b*a$ for all a, b in G .

CYCLIC GROUP: A group is cyclic if every element of G is a power a^k (k is an integer) of a fixed element $a \in G$. The element a is said to generate the group G or to be a generator of G . A cyclic group is always abelian and may be finite or infinite.

Rings

A ring R , sometimes denoted by $\{R, +, \times\}$, is a set of elements with two binary operations, called addition and multiplication, such that for all a, b, c in R the following axioms are obeyed

(A1–A5) R is an abelian group with respect to addition; that is, R satisfies axioms A1 through A5. For the case of an additive group, we denote the identity element as 0 and the inverse of a as $-a$.

(M1) Closure under multiplication: If a and b belong to R , then ab is also in R .

(M2) Associativity of multiplication: $a(bc) = (ab)c$ for all a, b, c in R .

(M3) Distributive laws: $a(b + c) = ab + ac$ for all a, b, c in R .
 $(a + b)c = ac + bc$ for all a, b, c in R .

A ring is said to be **commutative** if it satisfies the following additional condition:

(M4) Commutativity of multiplication: $ab = ba$ for all a, b in R .

Next, we define an integral domain, which is a commutative ring that obeys the following axioms

(M5) Multiplicative identity: There is an element 1 in R such that $a1 = 1a = a$ for all a in R .

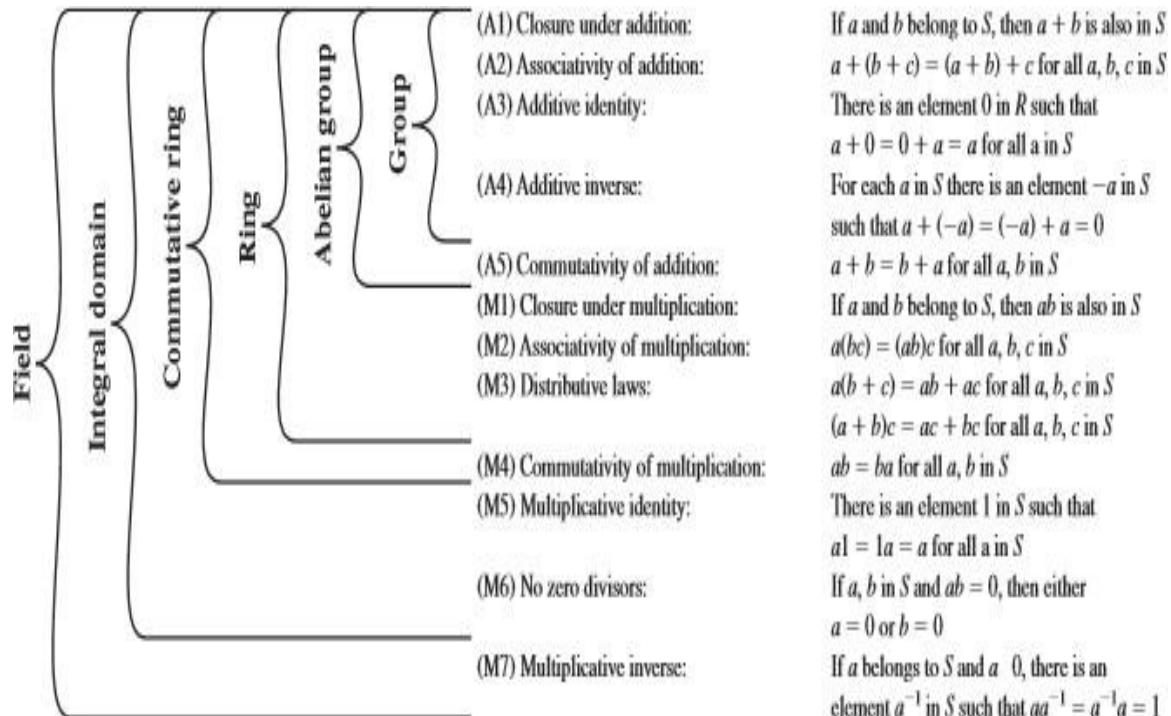
(M6) No zero divisors: If a, b in R and $ab = 0$, then either $a = 0$ or $b = 0$.

Fields

A field F , sometimes denoted by $\{F, +, \times\}$, is a set of elements with two binary operations, called addition and subtraction, such that for all a, b, c in F the following axioms are obeyed

(A1–M6) F is an integral domain; that is, F satisfies axioms A1 through A5 and M1 through M6.

(M7) Multiplicative inverse: For each a in F , except 0, there is an element a^{-1} in F such that $aa^{-1} = (a^{-1})a = 1$.

**Figure 2.2 Groups, Ring and Field**

2.2 MODULAR ARITHMETIC

If a is an integer and n is a positive integer, we define $a \bmod n$ to be the remainder when a is divided by n . The integer n is called the modulus. Thus, for any integer a , we can rewrite Equation as follows

$$a = qn + r \quad 0 \leq r < n; q = \lfloor a/n \rfloor$$

$$a = \lfloor a/n \rfloor \times n + (a \bmod n)$$

$$11 \bmod 7 = 4; \quad -11 \bmod 7 = 3$$

Two integers a and b are said to be **congruent modulo n** , if $(a \bmod n) = (b \bmod n)$. This is written as $a \equiv b \pmod{n}$.²

$$73 \equiv 4 \pmod{23}; \quad 21 \equiv -9 \pmod{10}$$

Note that if $a \equiv 0 \pmod{n}$, then $n|a$.

Modular Arithmetic Operations

A kind of integer arithmetic that reduces all numbers to one of a fixed set $[0, \dots, n-1]$ for some number n . Any integer outside this range is reduced to one in this range by taking the remainder after division by n .

Modular arithmetic exhibits the following properties

1. $[(a \text{ mod } n) + (b \text{ mod } n)] \text{ mod } n = (a + b) \text{ mod } n$
2. $[(a \text{ mod } n) - (b \text{ mod } n)] \text{ mod } n = (a - b) \text{ mod } n$
3. $[(a \text{ mod } n) \times (b \text{ mod } n)] \text{ mod } n = (a \times b) \text{ mod } n$

We demonstrate the first property. Define $(a \text{ mod } n) = r_a$ and $(b \text{ mod } n) = r_b$. Then we can write $a = r_a + jn$ for some integer j and $b = r_b + kn$ for some integer k . Then

$$\begin{aligned} (a + b) \text{ mod } n &= (r_a + jn + r_b + kn) \text{ mod } n \\ &= (r_a + r_b + (k+j)n) \text{ mod } n \\ &= (r_a + r_b) \text{ mod } n \\ &= [(a \text{ mod } n) + (b \text{ mod } n)] \text{ mod } n \end{aligned}$$

The remaining properties are proven as easily. Here are examples of the three properties:

Table 2.1 Arithmetic Modulo 8

$11 \text{ mod } 8 = 3; 15 \text{ mod } 8 = 7$							
$[(11 \text{ mod } 8) + (15 \text{ mod } 8)] \text{ mod } 8 = 10 \text{ mod } 8 = 2$							
$(11 + 15) \text{ mod } 8 = 26 \text{ mod } 8 = 2$							
$[(11 \text{ mod } 8) - (15 \text{ mod } 8)] \text{ mod } 8 = -4 \text{ mod } 8 = 4$							
$(11 - 15) \text{ mod } 8 = -4 \text{ mod } 8 = 4$							
$[(11 \text{ mod } 8) \times (15 \text{ mod } 8)] \text{ mod } 8 = 21 \text{ mod } 8 = 5$							
$(11 \times 15) \text{ mod } 8 = 165 \text{ mod } 8 = 5$							

+	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7	0
2	2	3	4	5	6	7	0	1
3	3	4	5	6	7	0	1	2
4	4	5	6	7	0	1	2	3
5	5	6	7	0	1	2	3	4
6	6	7	0	1	2	3	4	5
7	7	0	1	2	3	4	5	6

(a) Addition modulo 8

\times	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7
2	0	2	4	6	0	2	4	6
3	0	3	6	1	4	7	2	5
4	0	4	0	4	0	4	0	4
5	0	5	2	7	4	1	6	3
6	0	6	4	2	0	6	4	2
7	0	7	6	5	4	3	2	1

(b) Multiplication modulo 8

w	$-w$	w^{-1}
0	0	—
1	7	1
2	6	—
3	5	3
4	4	—
5	3	5
6	2	—
7	1	7

(c) Multiplicative inverses modulo 8

2.3 EUCLID'S ALGORITHM

One of the basic techniques of number theory is the Euclidean algorithm, which is a simple procedure for determining the greatest common divisor of two positive integers. First, we need a simple definition: Two integers are relatively prime if their only common positive integer factor is 1.

Greatest Common Divisor

Recall that nonzero b is defined to be a divisor of a if $a = mb$ for some m, where a,b, and m are integers. We will use the notation $\text{gcd}(a, b)$ to mean the greatest common divisor of a and b .The greatest common divisor of a and b is the largest integer that divides both a and b

.We also define $\text{gcd}(0,0) = 0$.

Algorithm

The Euclid's algorithm (or Euclidean Algorithm) is a method for efficiently finding the greatest common divisor (GCD) of two numbers. The GCD of two integers X and Y is the largest number that divides both of X and Y (without leaving a remainder).

For every non-negative integer, a and any positive integer b

$$\text{gcd}(a, b) = \text{gcd}(b, a \bmod b)$$

Algorithm Euclids (a, b)

$$\alpha = a$$

$$\beta = b$$

while ($\beta > 0$)

$$\text{Rem} = \alpha \bmod \beta$$

$$\alpha = \beta$$

$$\beta = \text{Rem}$$

return α

Steps for Another Method

$$a = q_1b + r_1; \quad 0 < r_1 < b$$

$$b = q_2r_1 + r_2; \quad 0 < r_2 < r_1$$

$$r_1 = q_3r_2 + r_3; \quad 0 < r_3 < r_2$$

$$r_{n-2} = q_n r_{n-1} + r_n; \quad 0 < r_n < r_{n-1}$$

$$r_{n-1} = q_1 r_n + 0$$

$$d = \text{gcd}(a, b) = r_n$$

Example 1:

$$\begin{aligned} \text{gcd}(55, 22) &= \text{gcd}(22, 55 \bmod 22) \\ &= \text{gcd}(22, 11) \\ &= \text{gcd}(11, 22 \bmod 11) \\ &= \text{gcd}(11, 0) \end{aligned}$$

$\text{gcd}(55, 22)$ is 11

Example 2:

$$\begin{aligned}
 \gcd(30, 50) &= \gcd(50, 30 \bmod 50) \\
 &= \gcd(50, 30) \\
 &= \gcd(30, 50 \bmod 30) \\
 &= \gcd(30, 20) \\
 &= \gcd(20, 30 \bmod 20) \\
 &= \gcd(20, 10) \\
 &= \gcd(10, 20 \bmod 10) \\
 &= \gcd(10, 0)
 \end{aligned}$$

$\gcd(30, 50)$ is 10

Another Method

To find $\gcd(30, 50)$

$$\begin{array}{lll}
 50 & = 1 \times 30 + 20 & \gcd(30, 20) \\
 30 & = 1 \times 20 + 10 & \gcd(20, 10) \\
 20 & = 1 \times 10 + 10 & \gcd(10, 10) \\
 10 & = 1 \times 10 + 0 & \gcd(10, 0)
 \end{array}$$

Therefore, $\gcd(30, 50) = 10$

Example 3:

$$\begin{aligned}
 \gcd(1970, 1066) &= \gcd(1066, 1970 \bmod 1066) \\
 &= \gcd(1066, 904) \\
 &= \gcd(904, 1066 \bmod 904) \\
 &= \gcd(904, 162) \\
 &= \gcd(162, 904 \bmod 162) \\
 &= \gcd(162, 94) \\
 &= \gcd(94, 162 \bmod 94) \\
 &= \gcd(94, 68) \\
 &= \gcd(68, 94 \bmod 68) \\
 &= \gcd(68, 26) \\
 &= \gcd(26, 68 \bmod 26) \\
 &= \gcd(26, 16) \\
 &= \gcd(16, 26 \bmod 16) \\
 &= \gcd(16, 10) \\
 &= \gcd(10, 16 \bmod 10) \\
 &= \gcd(10, 6) \\
 &= \gcd(6, 10 \bmod 6) \\
 &= \gcd(6, 4)
 \end{aligned}$$

$$\begin{aligned}
 &= \gcd(4, 6 \bmod 4) \\
 &= \gcd(4, 2) \\
 &= \gcd(2, 4 \bmod 2) \\
 &= \gcd(2, 0)
 \end{aligned}$$

$\gcd(1970, 1066)$ is 2

Another Method

To find $\gcd(1970, 1066)$

$$\begin{array}{lll}
 1970 & = 1 \times 1066 + 904 & \gcd(1066, 904) \\
 1066 & = 1 \times 904 + 162 & \gcd(904, 162) \\
 904 & = 5 \times 162 + 94 & \gcd(162, 94) \\
 162 & = 1 \times 94 + 68 & \gcd(94, 68) \\
 94 & = 1 \times 68 + 26 & \gcd(68, 26) \\
 68 & = 2 \times 26 + 16 & \gcd(26, 16) \\
 26 & = 1 \times 16 + 10 & \gcd(16, 10) \\
 16 & = 1 \times 10 + 6 & \gcd(10, 6) \\
 10 & = 1 \times 6 + 4 & \gcd(6, 4) \\
 6 & = 1 \times 4 + 2 & \gcd(4, 2) \\
 4 & = 2 \times 2 + 0 & \gcd(2, 0)
 \end{array}$$

Therefore, $\gcd(1970, 1066) = 2$

Extended Euclidean Algorithm

Extended Euclidean Algorithm is an efficient method of finding modular inverse of an integer.

Euclid's algorithm can be improved to give not just $\gcd(a, b)$, but also used to find the multiplicative inverse of a number with the modular value.

Example 1

Find the Multiplicative inverse of $17 \bmod 43$

$$17 \cdot 1 \bmod 43$$

$$17^* X = \bmod 43$$

$$X = 17 \cdot 1 \bmod 43$$

$$43 = 17 * 2 + 9$$

$$17 = 9 * 1 + 8$$

$$9 = 8 * 1 + 1$$

Rewrite the above equation

$$9 + 8(-1) = 1 \rightarrow (1)$$

$$17 + 9(-1) = 8 \rightarrow (2)$$

$$43 + 17(-2) = 9 \rightarrow (3)$$

Substitution

sub equ 2 in equ 1

$$(1) \rightarrow 9+8(-1) = 1 \quad [\text{Sub } 17+9(-1) = 8]$$

$$9+(17+9(-1))(-1) = 1$$

$$9+17(-1)+9(1)=1$$

$$17(-1)+9(2) = 1 \rightarrow (4)$$

Now sub equ (3) in equ (4)

$$43+17(-2) = 9 \rightarrow (3)$$

$$17(-1)+(43+17(-2))(2)=1$$

$$17(-1)+43(2)+17(-4)=1$$

$$17(-5)+43(2) = 1 \rightarrow (5)$$

Here -5 is the multiplicative inverse of 17. But inverse cannot be negative

$$17-1 \bmod 43 = -5 \bmod 43 = 38$$

So, 38 is the multiplicative inverse of 17.

Checking, $17^* X \equiv 1 \bmod 43$

$$17 * 38 \equiv 1 \bmod 43$$

$$646 \equiv 1 \bmod 43 \quad (15*43 = 645)$$

Example 2

Find the Multiplicative inverse of 1635 mod 26

$$1635-1 \bmod 26$$

$$1635 = 26 (62) + 23$$

$$26 = 23 (1) + 3$$

$$23 = 3(7) + 2$$

$$3 = 2(1) + 1$$

Rewriting the above equation

$$3+2(-1) = 1 \rightarrow (1)$$

$$23+3(-7) = 2 \rightarrow (2)$$

$$26+23(-1) = 3 \rightarrow (3)$$

$$1635+26(-62) = 23 \rightarrow (4)$$

Substitution

sub equ (2) in equ (1)

$$(2) \Rightarrow 23+3(-7) = 2$$

$$3+2(-1) = 1$$

$$3+(23+3(-7))(-1) = 1$$

$$3+23(-1)+3(7)=1$$

$$3(8)+23(-1) = 1 \rightarrow (5)$$

sub equ (3) in equ (5)

$$26+23(-1) = 3 \rightarrow (3)$$

$$(26+23(-1))(8) + 23 (-1) = 1$$

$$26(8) + 23 (-8) + 23 (-1) = 1$$

$$26 (8) + 23 (-9) = 1 \rightarrow (6)$$

Sub equ (4) in equ (6)

$$1635+26(-62) = 23 \rightarrow (4)$$

$$26 (8) + (1635 + 26 (-62)) (-9) = 1$$

$$26 (8) + 1635 (-9) + 26 (558) = 1$$

$$1635 (-9) + 26 (566) = 1 \rightarrow (7)$$

From equ (7) -9 is inverse of 1635, but negative cannot be inverse.

$$1635-1 \bmod 26 = -9 \bmod 26 = 17$$

So, the inverse of 1635 is 17.

Checking, $1635^* X \equiv 1 \bmod 26$

$$1635 * 17 \equiv 1 \bmod 26$$

$$27795 \equiv 1 \bmod 26 (1069*26 = 27794)$$

2.4 CONGRUENCE AND MATRICES

Properties of Congruences

Congruences have the following properties:

1. $a \equiv b \pmod{n}$ if $n|(a - b)$.
2. $a \equiv b \pmod{n}$ implies $b \equiv a \pmod{n}$.
3. $a \equiv b \pmod{n}$ and $b \equiv c \pmod{n}$ imply $a \equiv c \pmod{n}$.

To demonstrate the first point, if $n|(a - b)$, then $(a - b) = kn$ for some k . So we can write $a = b + kn$. Therefore, $(a \bmod n) = (\text{remainder when } b + kn \text{ is divided by } n) = (\text{remainder when } b \text{ is divided by } n) = (b \bmod n)$.

$23 \equiv 8 \pmod{5}$	because	$23 - 8 = 15 = 5 \times 3$
$-11 \equiv 5 \pmod{8}$	because	$-11 - 5 = -16 = 8 \times (-2)$
$81 \equiv 0 \pmod{27}$	because	$81 - 0 = 81 = 27 \times 3$

The remaining points are as easily proved.

Matrices

Matrix is a rectangular array in mathematics, arranged in rows and columns of numbers, symbols or expressions.

A matrix will be represented with their dimensions as $I \times m$ where I defines the row and m defines the columns

***m* columns**

$$\text{Rows} \left[\begin{array}{cccc} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \vdots & \vdots & & \vdots \\ a_{l1} & a_{l2} & \dots & a_{lm} \end{array} \right]$$

Examples of Matrices

1. Row Matrix
2. Column Matrix
3. Square Matrix
4. Zero Matrixes
5. Identity Matrix

$\begin{bmatrix} 2 & 1 & 5 & 11 \end{bmatrix}$ Row matrix	$\begin{bmatrix} 2 \\ 4 \\ 12 \end{bmatrix}$ Column matrix	$\begin{bmatrix} 23 & 14 & 56 \\ 12 & 21 & 18 \\ 10 & 8 & 31 \end{bmatrix}$ Square matrix	$\begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$ 0	$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ I
---------------------------------------------------------------------	----------------------------------------------------------------------	-----------------------------------------------------------------------------------------------------	---------------------------------------------------------------------	------------------------------------------------------------

2.5 FINITE FIELDS

FINITE FIELDS OF THE FORM GF(p)

The finite field of order is generally written ; GF stands for Galois field,in honor of the mathematician who first studied finite fields

Finite Fields of Order p

For a given prime, , we define the finite field of order , , as the set of integers together with the arithmetic operations modulo .

The simplest finite field is GF(2). Its arithmetic operations are easily summarized:

$+$	0	1
0	0	1
1	1	0

Addition

\times	0	1
0	0	0
1	0	1

Multiplication

w	$-w$	w^{-1}
0	0	—
1	1	1

Inverses

In this case, addition is equivalent to the exclusive-OR (XOR) operation, and multiplication is equivalent to the logical AND operation.

Finding the Multiplicative Inverse in It is easy to find the multiplicative inverse of an element in for small values of . You simply construct a multiplication table, such as shown in Table 2.2b, and the desired result can be read directly. However, for large values of , this approach is not practical. p p GF(p) GF(p)

Table 2.2 Arithmetic in GF(7)

+	0	1	2	3	4	5	6
0	0	1	2	3	4	5	6
1	1	2	3	4	5	6	0
2	2	3	4	5	6	0	1
3	3	4	5	6	0	1	2
4	4	5	6	0	1	2	3
5	5	6	0	1	2	3	4
6	6	0	1	2	3	4	5

(a) Addition modulo 7

×	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6
2	0	2	4	6	1	3	5
3	0	3	6	2	5	1	4
4	0	4	1	5	2	6	3
5	0	5	3	1	6	4	2
6	0	6	5	4	3	2	1

(b) Multiplication modulo 7

w	-w	w^{-1}
0	0	—
1	6	1
2	5	4
3	4	5
4	3	2
5	2	3
6	1	6

(c) Additive and multiplicative inverses modulo 7

2.5.1 Polynomial Arithmetic

We are concerned with polynomials in a single variable and we can distinguish three classes of polynomial arithmetic.

- Ordinary polynomial arithmetic, using the basic rules of algebra.
- Polynomial arithmetic in which the arithmetic on the coefficients is performed modulo ;that is, the coefficients are in .

Polynomial arithmetic in which the coefficients are in , and the polynomials are defined modulo a polynomial whose highest power is some integer .

Ordinary Polynomial Arithmetic

A polynomial of degree (integer) is an expression of the form

A **polynomial** of degree n (integer $n \geq 0$) is an expression of the form

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

where the a_i are elements of some designated set of numbers S , called the **coefficient set**, and $a_n \neq 0$. We say that such polynomials are defined over the coefficient set S .

A zero-degree polynomial is called a **constant polynomial** and is simply an element of the set of coefficients. An n th-degree polynomial is said to be a **monic polynomial** if $a_n = 1$.

In the context of abstract algebra, we are usually not interested in evaluating a polynomial for a particular value of x [e.g., $f(7)$]. To emphasize this point, the variable x is sometimes referred to as the **indeterminate**.

Addition and subtraction are performed by adding or subtracting corresponding coefficients. Thus, if

$$f(x) = \sum_{i=0}^n a_i x^i; \quad g(x) = \sum_{i=0}^m b_i x^i; \quad n \geq m$$

then addition is defined as

$$f(x) + g(x) = \sum_{i=0}^m (a_i + b_i)x^i + \sum_{i=m+1}^n a_i x^i$$

and multiplication is defined as

$$f(x) \times g(x) = \sum_{i=0}^{n+m} c_i x^i$$

where

$$c_k = a_0 b_k + a_1 b_{k-1} + \cdots + a_{k-1} b_1 + a_k b_0$$

As an example, let $f(x) = x^3 + x^2 + 2$ and $g(x) = x^2 - x + 1$, where S is the set of integers. Then

$$\begin{aligned} f(x) + g(x) &= x^3 + 2x^2 - x + 3 \\ f(x) - g(x) &= x^3 + x + 1 \\ f(x) \times g(x) &= x^5 + 3x^2 - 2x + 2 \end{aligned}$$

Polynomial Arithmetic with Coefficients in

Let us now consider polynomials in which the coefficients are elements of some field F; we refer to this as a polynomial over the field F. In that case, it is easy to show that the set of such polynomials is a ring, referred to as a polynomial ring. That is, if we consider each distinct polynomial to be an element of the set, then that set is a ring 8 when polynomial arithmetic is performed on polynomials over a field, then division is possible. Note that this does not mean that exact division is possible. Let us clarify this distinction. Within a field, given two elements and, the quotient is also an element of the field. However, given a ring that is not a field, in Ra /b ba Zp

$$\begin{array}{r} x^3 + x^2 + 2 \\ + (x^2 - x + 1) \\ \hline x^3 + 2x^2 - x + 3 \end{array}$$

(a) Addition

$$\begin{array}{r} x^3 + x^2 + 2 \\ - (x^2 - x + 1) \\ \hline x^3 + x + 1 \end{array}$$

(b) Subtraction

$$\begin{array}{r} x^3 + x^2 + 2 \\ \times (x^2 - x + 1) \\ \hline x^3 + x^2 + 2 \\ - x^4 - x^3 - 2x \\ \hline x^5 + x^4 + 2x^2 \\ \hline x^5 + 3x^2 - 2x + 2 \end{array}$$

(c) Multiplication

$$\begin{array}{r} x+2 \\ \hline x^2 - x + 1 \sqrt{x^3 + x^2 + 2} \\ x^3 - x^2 + x \\ \hline 2x^2 - x + 2 \\ 2x^2 - 2x + 2 \\ \hline x \end{array}$$

(d) Division

Figure 2.3 Examples of Polynomial Arithmetic

A polynomial over a field is called irreducible if and only if cannot be expressed as a product of two polynomials, both over, and both of degree lower than that of. By analogy to integers, an irreducible polynomial is also called a prime polynomial.

2.6 SYMMETRIC KEY CIPHERS

Symmetric ciphers use the same cryptographic keys for both encryption of plaintext and decryption of ciphertext. They are faster than asymmetric ciphers and allow encrypting large sets of data. However, they require sophisticated mechanisms to securely distribute the secret keys to both parties

Definition

A symmetric cipher defined over (K, M, C) , where:

- K - a set of all possible keys,
- M - a set of all possible messages,
- C - a set of all possible ciphertexts

is a pair of efficient algorithms (E, D) , where:

- $E: K \times M \rightarrow C$
- $D: K \times C \rightarrow M$

such that for every m belonging to M , k belonging to K there is an equality:

- $D(k, E(k, m)) = m$ (the consistency rule)

→ Function E is often randomized

→ Function D is always deterministic

Types of keys are used in symmetric key cryptography

Symmetric encryption (figure 2.4) uses a single key that needs to be shared among the people who need to receive the message while asymmetrical encryption uses a pair of public key and a private key to encrypt and decrypt messages when communicating.

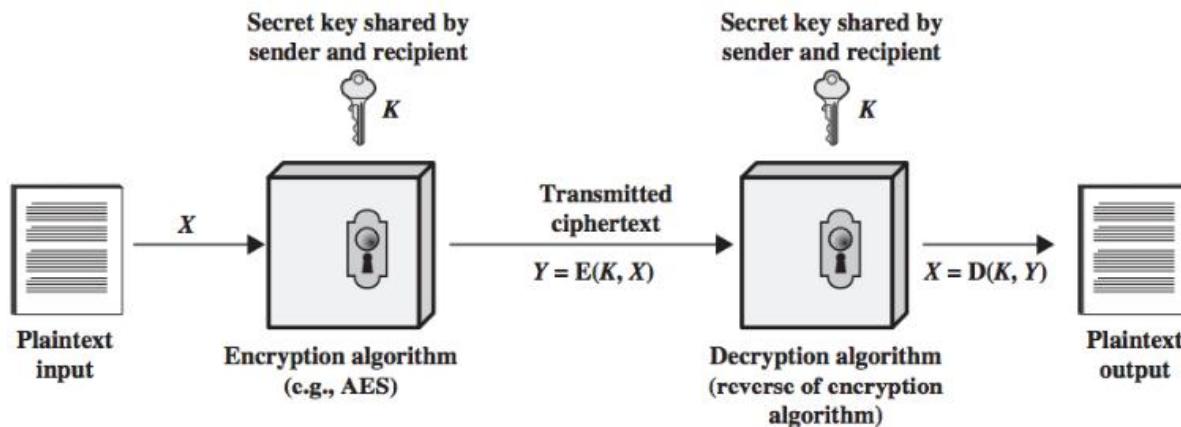


Figure 2.4 Simplified Model of Symmetric Encryption

2.7 SIMPLIFIED DATA ENCRYPTION STANDARD (S-DES)

The overall structure of the simplified DES shown in Figure 2.5. The S-DES encryption algorithm takes an 8-bit block of plaintext (example: 10111101) and a 10-bit key as input and produces an 8-bit block of ciphertext as output.

The S-DES decryption algorithm takes an 8-bit block of ciphertext and the same 10-bit key used to produce that ciphertext as input and produces the original 8-bit block of plaintext.

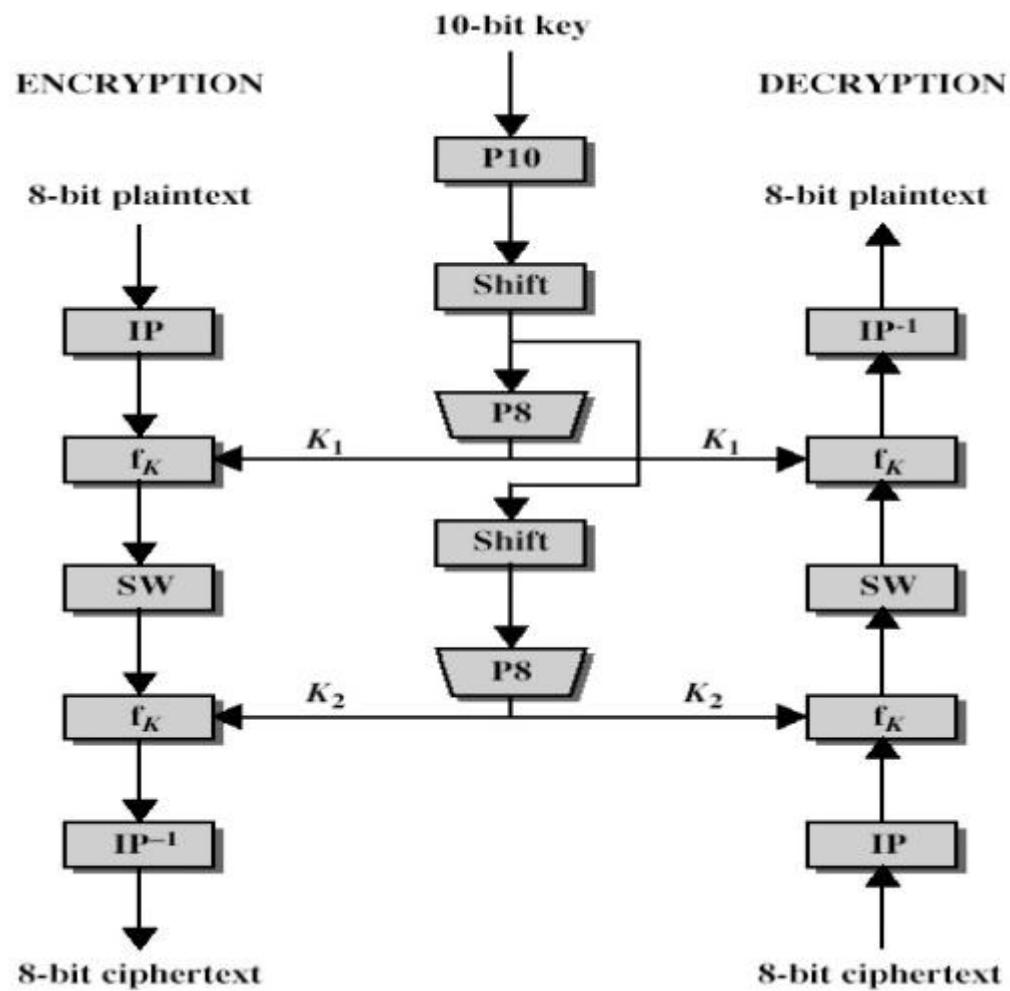


Figure 2.5 Overview of S-DES Algorithm

The encryption algorithm involves five functions:

- An initial permutation (IP)
- A complex function labeled f_K , which involves both permutation and substitution operations and depends on a key input.
- A simple permutation function that switches (SW) the two halves of the data.
- The function f_K again.

A permutation function that is the inverse of the initial permutation

The function f_K takes as input not only the data passing through the encryption algorithm, but also an 8-bit key. Here a 10-bit key is used from which two 8-bit subkeys are generated.

The key is first subjected to a permutation (P10). Then a shift operation is performed. The output of the shift operation then passes through a permutation function that produces an 8-bit output (P8) for the first subkey (K_1).

The output of the shift operation also feeds into another shift and another instance of P8 to produce the second subkey (K_2).

The encryption algorithm can be expressed as a composition of functions:

$IP-1 \circ fK2 \circ SW \circ fk1 \circ IP$, which can also be written as

Ciphertext = $IP-1(fK2(SW(fk1(IP(\text{plaintext}))))))$

Where

$K1 = P8(\text{Shift}(P10(\text{Key})))$

$K2 = P8(\text{Shift}(\text{shift}(P10(\text{Key}))))$

Decryption can be shown as Plaintext = $IP-1(fK1(SW(fk2(IP(\text{ciphertext}))))))$

2.7.1 S-DES Key Generation

S-DES depends on the use of a 10-bit key shared between sender and receiver.

From this key, two 8-bit subkeys are produced for use in particular stages of the encryption and decryption algorithm.(Figure 2.6)

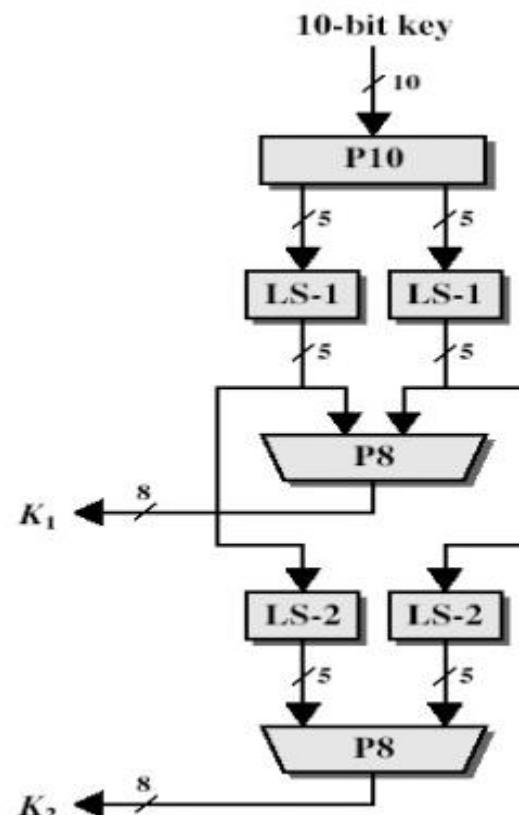


Figure 2.6 S-DES Key Generation

First, permute the key in the following fashion. Let the 10-bit key be designated as $(k_1, k_2, k_3, k_4, k_5, k_6, k_7, k_8, k_9, k_{10})$. Then the permutation $P10$ is defined as:

$P10(k_1, k_2, k_3, k_4, k_5, k_6, k_7, k_8, k_9, k_{10}) = (k_3, k_5, k_2, k_7, k_4, k_{10}, k_{10}, k_1, k_9, k_8, k_6)$.

$P10$ can be concisely defined by the display:

P10									
3	5	2	7	4	10	1	9	8	6

This table is read from left to right; each position in the table gives the identity of the input bit that produces the output bit in that position. So, the first output bit is bit 3 of the input; the second output bit is bit 5 of the input, and so on.

Example

The 10 bit key is (1010000010), now find the permutation from P10 for this key so it becomes (10000 01100).

Next, perform a circular left shift (LS-1), or rotation, separately on the first five bits and the second five bits. In our example, the result is (00001 11000).

Next, apply P8, which picks out and permutes 8 of the 10 bits according to the following rule:

P8							
6	3	7	4	8	5	10	9

So, The result is subkey 1 (K1). In our example, this yield (10100100).

Then go back to the pair of 5-bit strings produced by the two LS-1 functions and performs a circular left shift of 2 bit positions on each string. In our example, the value (00001 11000) becomes (00100 00011).

Finally, P8 is applied again to produce K2. In our example, the result is (01000011).

2.7.2 S-DES Encryption

Encryption involves the sequential application of five functions (Figure 2.7).

1. Initial Permutations

The input to the algorithm is an 8-bit block of plaintext, which we first permute using the IP function

IP							
2	6	3	1	4	8	5	7

The plaintext is 10111101

Permutated output is 01111110

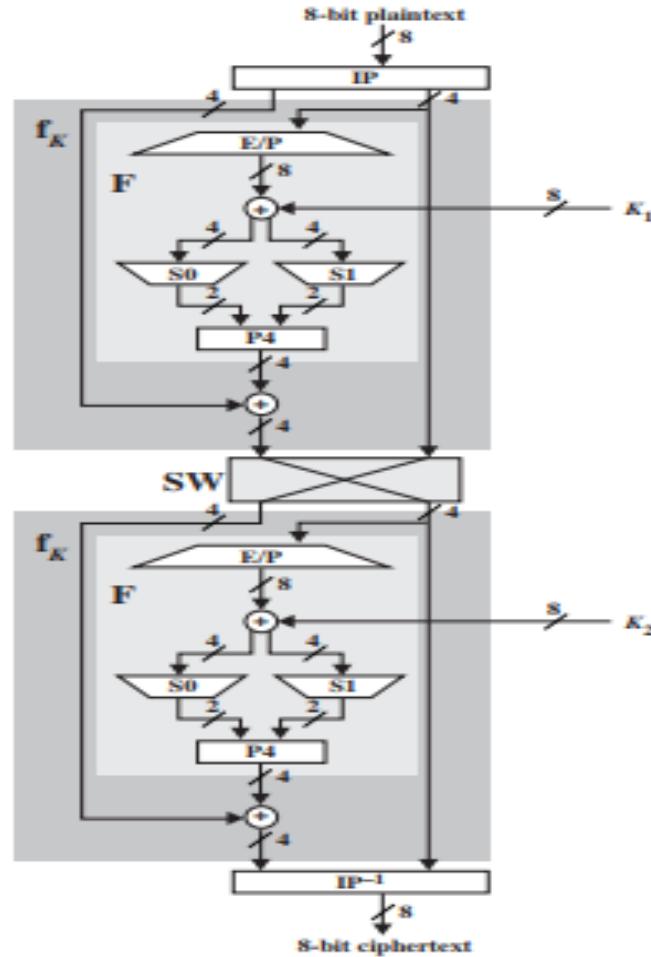


Figure 2.7 S-DES Encryption

2. The Function f_k

The most complex component of S-DES is the function f_k , which consists of a combination of permutation and substitution functions. The functions can be expressed as follows. Let L and R be the leftmost 4 bits and rightmost 4 bits of the 8-bit input to f_k , and let F be a mapping (not necessarily one to one) from 4-bit strings to 4-bit strings. Then we let

$$f_k(L, R) = (L \oplus F(R, SK), R)$$

Where SK is a sub key and \oplus is the bit-by-bit exclusive OR function

Now, describe the mapping F. The input is a 4-bit number (n₁ n₂ n₃ n₄). The first operation is an expansion/permuation operation:

E/P							
4	1	2	3	2	3	4	1

Now, find the E/P from IP

IP = 01111110, it becomes

E/P = 01111101

Now, XOR with K1

$$\Rightarrow 01111101 \oplus 10100100 = 11011001$$

The first 4 bits (first row of the preceding matrix) are fed into the S-box S0 to produce a 2-bit output, and the remaining 4 bits (second row) are fed into S1 to produce another 2-bit output.

These two boxes are defined as follows:

$$S_0 = \begin{matrix} & 0 & 1 & 2 & 3 \\ 0 & \begin{bmatrix} 1 & 0 & 3 & 2 \\ 3 & 2 & 1 & 0 \end{bmatrix} & & \\ 1 & \begin{bmatrix} 0 & 2 & 1 & 3 \\ 3 & 1 & 3 & 2 \end{bmatrix} & & \\ 2 & & & \\ 3 & & & \end{matrix} \quad S_1 = \begin{matrix} & 0 & 1 & 2 & 3 \\ 0 & \begin{bmatrix} 0 & 1 & 2 & 3 \\ 2 & 0 & 1 & 3 \end{bmatrix} & & \\ 1 & \begin{bmatrix} 3 & 0 & 1 & 0 \\ 2 & 1 & 0 & 3 \end{bmatrix} & & \\ 2 & & & \\ 3 & & & \end{matrix}$$

The S-boxes operate as follows. The first and fourth input bits are treated as a 2-bit number that specify a row of the S-box, and the second and third input bits specify a column of the S-box. Each S box gets 4-bit input and produce 2 bits as output. It follows 00-0, 01-1, 10-2, 11-3 scheme.

Here, take first 4 bits,

$$S_0 \Rightarrow 1101$$

$$\left. \begin{array}{l} 11 \rightarrow 3 \\ 10 \rightarrow 2 \end{array} \right\} \Rightarrow 3 \Rightarrow 11$$

So, we get 1110

➤ Now, find P₄

$$S_1 \Rightarrow 1001$$

$$\left. \begin{array}{l} 11 \rightarrow 3 \\ 00 \rightarrow 0 \Rightarrow 2 \Rightarrow 1 \end{array} \right\} 0$$

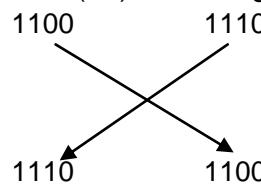
P ₄			
2	4	3	1

After P₄, the value is 1011

Now, XOR operation 1011 ⊕ 0111 => 1100

3. The Switch function

- The switch function (sw) interchanges the left and right 4 bits.



4. Second function f_k

- First, do E/P function and XOR with K₂, the value is 01101001 ⊕ 01000011, the answer is 00101010

- Now, find S₀ and S₁

$$S_0 \Rightarrow \begin{matrix} 00 \rightarrow 0 \\ 01 \rightarrow 1 \end{matrix} \Rightarrow 0 \quad S_1 \Rightarrow \begin{matrix} 10 \rightarrow 2 \\ 01 \rightarrow 1 \end{matrix} = > 0 \Rightarrow 00$$

Value is 0000

- Now, find P₄ and XOR operation

After P₄ => 0000 ⊕ 1110 = 1110, then concatenate last 4 bits after interchange in sw.

- Now value is 11101100

5. Find IP⁻¹

IP -1							
4	1	3	5	7	2	8	6

So, value is 01110101

The Ciphertext is 01110101

2.8.3 S-DES Decryption

- Decryption involves the sequential application of five functions.

1. Find IP

- After IP, value is 11101100

2. Function f_k

- After step 2, the answer is 11101100

3. Swift

- The answer is 11001110

4. Second f_k

- The answer is 01111110

5. Find IP-1

- 101111101 → Plaintext

2.8 DATA ENCRYPTION STANDARD

The most widely used encryption scheme is based on the Data Encryption Standard (DES) adopted in 1977. The algorithm itself is referred to as the Data Encryption Algorithm (DEA).

For DES, data are encrypted in 64-bit blocks using a 56-bit key. The algorithm transforms 64-bit input in a series of steps into a 64-bit output.

2.8.1 DES Encryption

The overall scheme for DES encryption is illustrated in the Figure 2.8. There are two inputs to the encryption function: the **plaintext** to be encrypted and the **key**. The plaintext must be 64 bits in length and the key is 56 bits in length.

2.8.2 General Depiction of DES Encryption Algorithm**Phase 1**

Looking at the left-hand side of the figure 2.8, we can see that the processing of the plaintext proceeds in three phases.

First, the 64-bit plaintext passes through an initial permutation (IP) that rearranges the bits to produce the *permuted input*.

Phase 2:

This is followed by a phase consisting of 16 rounds of the same function, which involves both permutation and substitution functions.

The output of the last (sixteenth) round consists of 64 bits that are a function of the input plaintext and the key. The left and right halves of the output are swapped to produce the preoutput.

Phase 3:

Finally, the preoutput is passed through a permutation (IP^{-1}) that is the inverse of the initial permutation function, to produce the 64-bit ciphertext.

The right-hand portion of Figure shows the way in which the 56-bit key is used.

Operation on key:

Initially, the key is passed through a permutation function. Then, for each of the 16 rounds, a *subkey* (K_i) is produced by the combination of a left circular shift and a permutation. The permutation function is the same for each round, but a different subkey is produced because of the repeated shifts of the key bits.

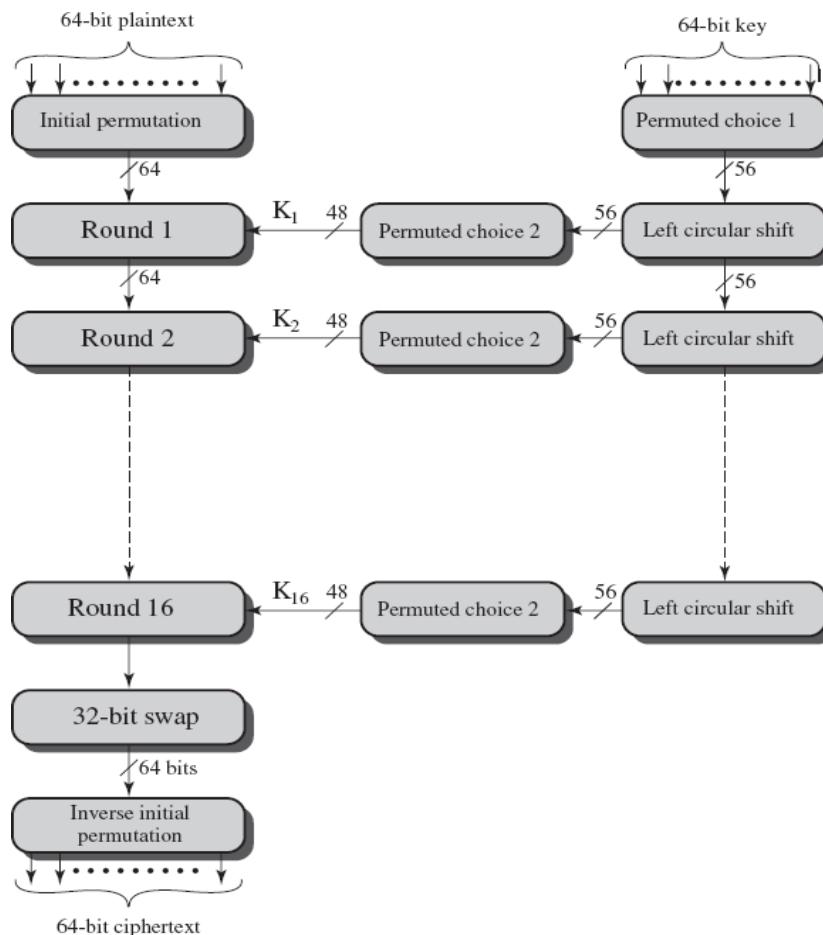


Figure 2.8 DES Encryption Algorithm

Initial Permutation

The input to a table consists of 64 bits numbered from 1 to 64. The 64 entries in the permutation table contain a permutation of the numbers from 1 to 64. Each entry in the permutation table indicates the position of a numbered input bit in the output, which also consists of 64 bits.

Permutation Tables for DES**(a) Initial Permutation (IP)**

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

Inverse Initial Permutation (IP⁻¹)

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

Expansion Permutation (E)

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

Permutation Function (P)

16	7	20	21	29	12	28	17
1	15	23	26	5	18	31	10
2	8	24	14	32	27	3	9
19	13	30	6	22	11	4	25

Consider the following 64-bit input M:

M1	M2	M3	M4	M5	M6	M7	M8
M9	M10	M11	M12	M13	M14	M15	M16
M17	M18	M19	M20	M21	M22	M23	M24
M25	M26	M27	M28	M29	M30	M31	M32
M33	M34	M35	M36	M37	M38	M39	M40
M41	M42	M43	M44	M45	M46	M47	M48
M49	M50	M51	M52	M53	M54	M55	M56
M57	M58	M59	M60	M61	M62	M63	M64

where M_i is a binary digit. Then the permutation $X = IP(M)$ is as follows:

M_{58}	M_{50}	M_{42}	M_{34}	M_{26}	M_{18}	M_{10}	M_2
M_{60}	M_{52}	M_{44}	M_{36}	M_{28}	M_{20}	M_{12}	M_4
M_{62}	M_{54}	M_{46}	M_{38}	M_{30}	M_{22}	M_{14}	M_6
M_{64}	M_{56}	M_{48}	M_{40}	M_{32}	M_{24}	M_{16}	M_8
M_{57}	M_{49}	M_{41}	M_{33}	M_{25}	M_{17}	M_9	M_1
M_{59}	M_{51}	M_{43}	M_{35}	M_{27}	M_{19}	M_{11}	M_3
M_{61}	M_{53}	M_{45}	M_{37}	M_{29}	M_{21}	M_{13}	M_5
M_{63}	M_{55}	M_{47}	M_{39}	M_{31}	M_{23}	M_{15}	M_7

Inverse permutation $Y = IP^{-1}(X) = IP^{-1}(IP(M))$, Therefore we can see that the original ordering of the bits is restored.

2.8.3 Details of Single Round

The below figure 2.9 shows the internal structure of a single round. The left and right halves of each 64-bit intermediate value are treated as separate 32-bit quantities, labeled L (left) and R (right). The overall processing at each round can be summarized in the following formulas:

$$L_i = R_{i-1}$$

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

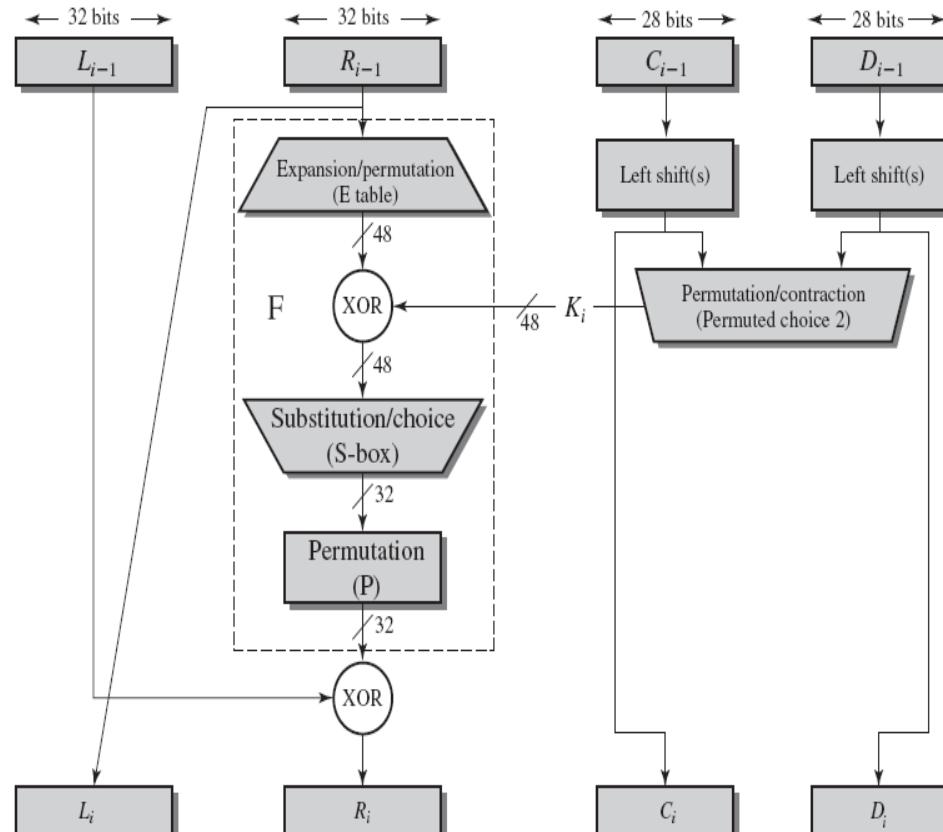


Figure 2.9 Single Round of DES Algorithm

The round key K_i is 48 bits. The R input is 32 bits. This R input is first expanded to 48 bits by using a table that defines a permutation plus an expansion that involves duplication of 16 of the R bits. The resulting 48 bits are XORed with K_i . This 48-bit result passes through a substitution function that produces a 32-bit output, which is then permuted.

Definition of S-Boxes

The substitution consists of a set of eight S-boxes, each of which accepts 6 bits as input and produces 4 bits as output. The first and last bits of the input to box S_i form a 2-bit binary number to select one of four substitutions defined by the four rows in the table for S_i . The middle four bits select one of the sixteen columns as shown in figure 2.10.

The decimal value in the cell selected by the row and column is then converted to its 4-bit representation to produce the output.

For example, in S_1 for input 011001, the row is 01 (row 1) and the column is 1100 (column 12). The value in row 1, column 12 is 9, so the output is 1001.

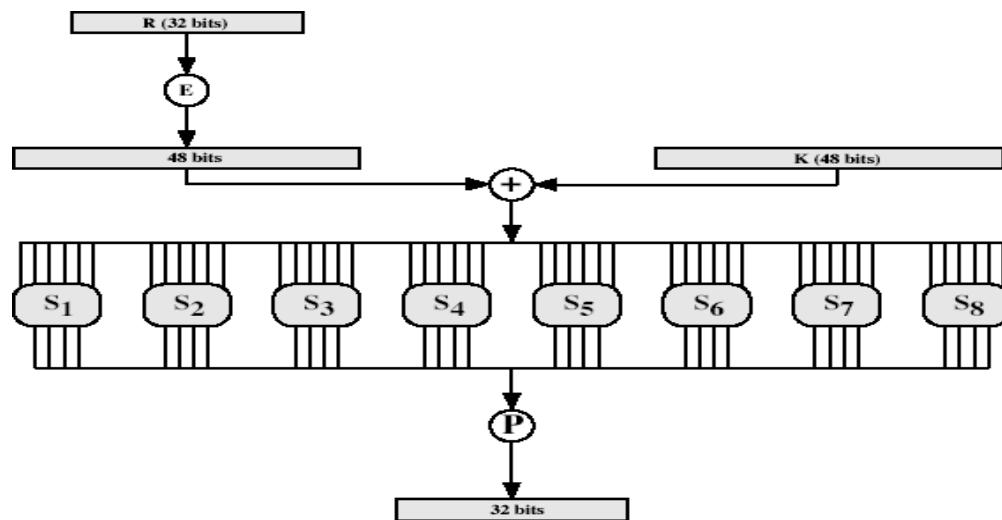


Fig 2.10 Calculation of $F(R, K)$

2.8.4 Key Generation

The 64-bit key is used as input to the algorithm. The bits of the key are numbered from 1 through 64; every eighth bit is ignored. The key is first subjected to a permutation governed by a table labeled Permuted Choice One. The resulting 56-bit key is then treated as two 28-bit quantities, labeled C_0 and D_0 .

At each round, C_{i-1} and D_{i-1} are separately subjected to a circular left shift, or rotation, of 1 or 2 bits. These shifted values serve as input to the next round. They also serve as input to Permuted Choice 2, which produces a 48-bit output that serves as input to the function $F(R_{i-1}, K_i)$.

DES Key Schedule Calculation

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

(b) Permuted Choice One (PC-1)

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

(c) Permuted Choice Two (PC-2)

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

(d) Schedule of Left Shifts

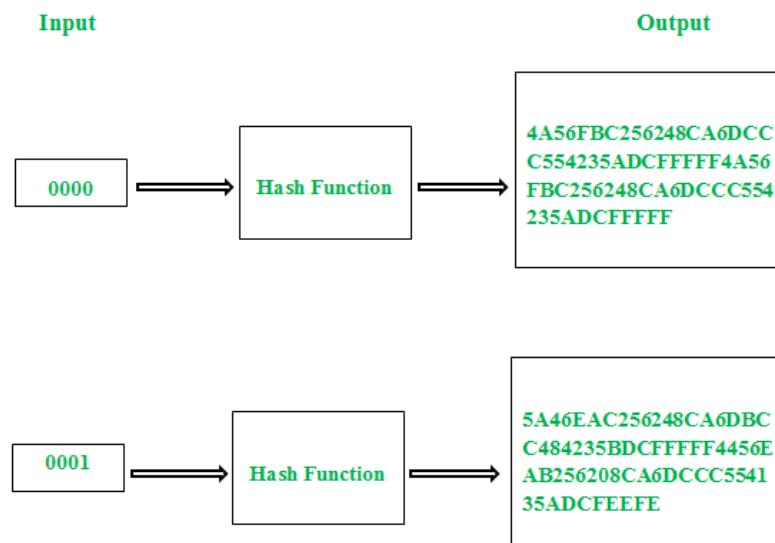
Roundnumber: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
 Bits rotated : 1 1 2 2 2 2 2 2 1 2 2 2 2 2 2 1

2.8.5 DES Decryption:

As with any Feistel cipher, decryption uses the same algorithm as encryption, except that the application of the subkeys is reversed. Additionally, the initial and final permutations are reversed.

2.8.6 The Avalanche Effect:

A desirable property of any encryption algorithm is that a small change in either the plaintext or the key should produce a significant change in the ciphertext. In particular, a change in one bit of the plaintext or one bit of the key should produce a change in many bits of the ciphertext.



2.9 THE STRENGTH OF DES

The strength of DES depends on two factors: **key size** and the **nature of the algorithm**.

1. The Use of 56-Bit Keys

With a key length of 56 bits, there are 2^{56} possible keys, which is approximately 7.2×10^{16} . Thus, a brute-force attack appears impractical.

2. The Nature of the DES Algorithm

In DES algorithm, eight substitution boxes called S-boxes that are used in each iteration. Because the design criteria for these boxes, and indeed for the entire algorithm, were not made public, there is a suspicion that the boxes were constructed in such a way that cryptanalysis is possible for an opponent who knows the weaknesses in the S-boxes. Despite this, no one has so far succeeded in discovering the supposed fatal weaknesses in the S-boxes.

3. Timing Attacks

A timing attack is one in which information about the key or the plaintext is obtained by observing how long it takes a given implementation to perform decryptions on various ciphertexts. A timing attack exploits the fact that an encryption or decryption algorithm often takes slightly different amounts of time on different inputs.

2.9.1 Attacks on DES:

Two approaches are:

1. Differential crypt analysis
2. Linear crypt analysis

2.9.1.1 Differential Cryptanalysis

Differential cryptanalysis is the first published attack that is capable of breaking DES in less than 255 complexities. The need to strengthen DES against attacks using differential cryptanalysis played a large part in the design of the S-boxes and the permutation P.

- One of the most significant recent (public) advances in cryptanalysis
- Powerful method to analyze block ciphers
- Used to analyze most current block ciphers with varying degrees of success

Differential Cryptanalysis Attack:

The differential cryptanalysis attack is complex. The rationale behind differential cryptanalysis is to observe the behavior of pairs of text blocks evolving along each round of the cipher, instead of observing the evolution of a single text block.

Consider the original plaintext block m to consist of two halves m_0, m_1 . Each round of DES maps the right-hand input into the left-hand output and sets the right-hand output to be a function of the left-hand input and the subkey for this round.

So, at each round, only one new 32-bit block is created. If we label each new block m_i ($2 \leq i \leq 17$), then the intermediate message halves are related as follows:

$$m_{i+1} = m_{i-1} \oplus f(m_i, K_i), i = 1, 2, \dots, 16$$

In differential cryptanalysis, we start with two messages, m and m' , with a known XOR difference $\Delta m = m \oplus m'$, and consider the difference between the intermediate message halves: $m_i = m_i \oplus m'_i$. Then we have:

$$\begin{aligned} \Delta m_{i+1} &= m_{i+1} \oplus m''_{i-1} \\ &= [m_{i-1} \oplus f(m_i, k_i)] \oplus [m''_{i-1} \oplus f(m'', i, k_i)] \\ &= \Delta m_{i-1} \oplus [f(m_i, k_i) \oplus f(m'', i, k_i)] \end{aligned}$$

Let us suppose that there are many pairs of inputs to f with the same difference yield the same output difference if the same subkey is used.

Therefore, if we know Δm_{i+1} and Δm_i with high probability, then we know Δm_{i-1} with high probability. Furthermore, if a number of such differences are determined, it is feasible to determine the subkey used in the function f .

2.9.1.2 Linear Cryptanalysis

This attack is based on the fact that linear equation can be framed to describe the transformations.

The principle of linear crypt analysis is as follows

Length of CT and PT = n bits;

key=mbit

Block of cipher text is $c[1]c[2]\dots c[n]$; Block of key is $k[1]k[2]\dots k[m]$

$$A[i, j, \dots, k] = A[i] \oplus A[j] \oplus \dots \oplus A[k]$$

- Can attack DES with 247 known plaintexts, still in practice infeasible
- Find linear approximations with prob $p \neq \frac{1}{2}$
- $P[i_1, i_2, \dots, i_a](+)c[j_1, j_2, \dots, j_b] = k[k_1, k_2, \dots, k_c]$ Where i_a, j_b, k_c are bit locations in p, c, k

2.10 BLOCK CIPHER PRINCIPLES

There are three critical aspects of block cipher design:

1. Number of rounds,
2. Design of the function F
3. Key scheduling.

Number of Rounds

- When the greater the number of rounds, the more difficult it is to perform cryptanalysis, even for a relatively weak F.
- The number of rounds is chosen so that known cryptanalytic efforts require greater effort than a simple brute-force key search attack
- When round DES S= 16, a differential cryptanalysis attack is slightly less efficient than brute force, the differential cryptanalysis attack requires 2^{55} operations.
- It makes it easy to judge the strength of an algorithm and to compare different algorithms.

Design of Function F

This is the most important function

Criteria needed for F,

- It must be difficult to “unscramble” the substitution performed by F.
- The function should satisfy **strict avalanche criterion (SAC)** which states that any output bit j of an S-box should change with probability 1/2 when any single input bit i is inverted for all i, j .
- The function should satisfy **bit independence criterion(BIC)**, which states that output bits j and k should change independently when any single input bit i is inverted for all i, j , and k .

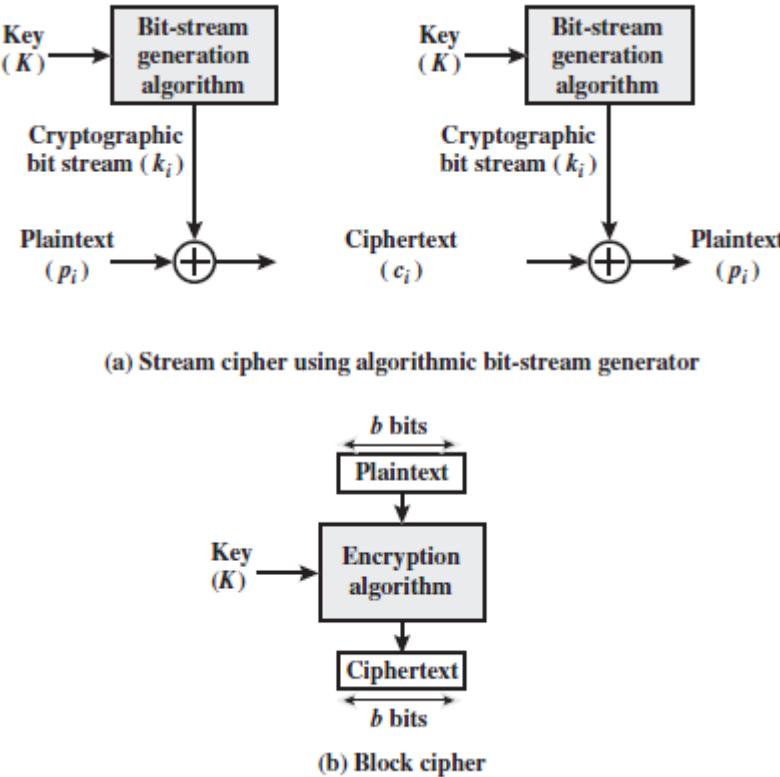
Key Schedule Algorithm

- The key is used to generate one sub key for each round.
- The sub keys to maximize the difficulty of deducing individual sub keys and the difficulty of working back to the main key.

2.10.1 Stream Cipher and Block Cipher

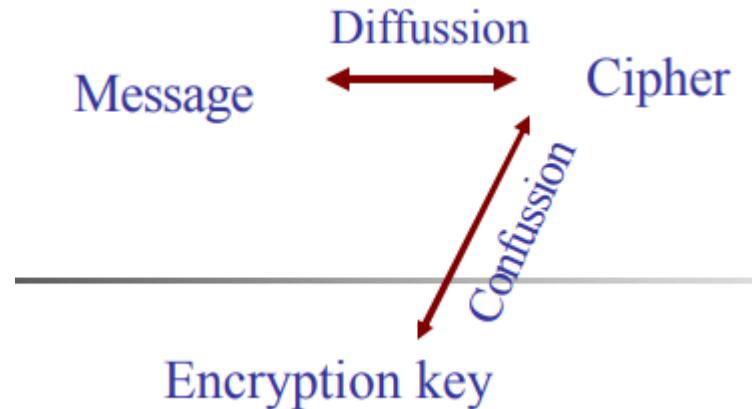
A stream cipher is one that encrypts a digital data stream one bit or one byte at a time.
E.g, vigenere cipher. Figure (2.11a)

A block cipher is one in which a block of plaintext is treated as a whole and used to produce a cipher text block of equal length. Typically, a block size of 64 or 128 bits is used. Figure (2.11b)

**Figure 2.11 Stream Cipher and Block Cipher**

- Many block ciphers have a Feistel structure. Such a structure consists of a number of identical rounds of processing.
- In each round, a substitution is performed on one half of the data being processed, followed by a permutation that interchanges the two halves.
- The original key is expanded so that a different key is used for each round.
- The Data Encryption Standard (DES) has been the most widely used encryption algorithm. It exhibits the classic Feistel structure.
- The DES uses a 64-bit block and a 56-bit key. Two important methods of cryptanalysis are differential cryptanalysis and linear cryptanalysis. DES has been shown to be highly resistant to these two types of attack.
- A block cipher operates on a plaintext block of n bits to produce a ciphertext block of n bits. There are possible different plaintext blocks and, for the encryption to be reversible (i.e., for decryption to be possible), each must produce a unique ciphertext block. Such a transformation is called reversible, or non singular
- In particular, Feistel proposed the use of a cipher that alternates substitutions and permutations, where these terms are defined as follows:
 - **Substitution:** Each plaintext element or group of elements is uniquely replaced by a corresponding ciphertext element or group of elements.
 - **Permutation:** A sequence of plaintext elements is replaced by a permutation of that sequence. That is, no elements are added or deleted or replaced in the sequence, rather the order in which the elements appear in the sequence is changed.

- Two methods for frustrating statistical cryptanalysis are:
 - **Diffusion** – Each plaintext digit affects many ciphertext digits, or each ciphertext digit is affected by many plaintext digits.
 - **Confusion** – Make the statistical relationship between a plaintext and the corresponding ciphertext as complex as possible in order to thwart attempts to deduce the key.



2.10.2 Feistel cipher structure

- The left-hand side of figure 2.12 depicts the structure proposed by Feistel.
- The input to the encryption algorithm is a plaintext block of length $2w$ bits and a key K . the plaintext block is divided into two halves L_0 and R_0 .
- The two halves of the data pass through n rounds of processing and then combine to produce the ciphertext block. Each round i has inputs L_{i-1} and R_{i-1} , derived from the previous round, as well as the subkey K_i , derived from the overall key K .
- In general, the subkeys K_i are different from K and from each other. All rounds have the same structure.
- A substitution is performed on the left half of the data (as similar to S-DES). This is done by applying a round function F to the right half of the data and then taking the XOR of the output of that function and the left half of the data.
- The round function has the same general structure for each round but is parameterized by the round subkey k_i . Following this substitution, a permutation is performed that consists of the interchange of the two halves of the data.
- This structure is a particular form of the substitution-permutation network.

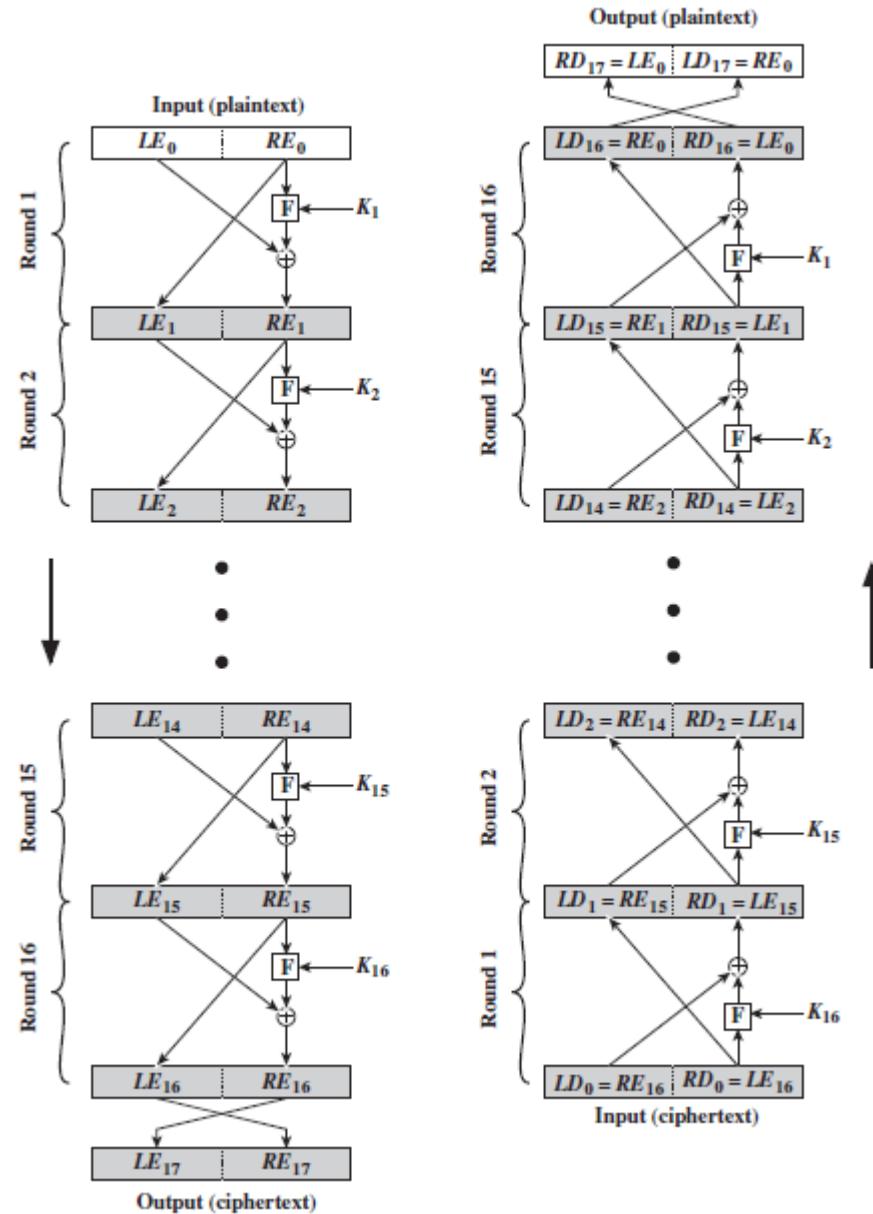


Figure 2.12 Feistel Encryption and Decryption (16 rounds)

The features of Feistel network are:

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

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

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

$$\begin{aligned} LE_{16} &= RE_{15} \\ RE_{16} &= LE_{15} \oplus F(LE_{15}, K_{16}) \end{aligned}$$

On the decryption side,

$$\begin{aligned} LD_1 &= RD_0 = LE_{16} = RE_{15} \\ RD_1 &= LD_0 \oplus F(RD_0, K_{16}) \\ &= RE_{16} \oplus F(RE_{15}, K_{16}) \\ &= [LE_{15} \oplus F(LE_{15}, K_{16})] \oplus F(LE_{15}, K_{16}) \\ &= LE_{15} \end{aligned}$$

Therefore, $LD_1 = RE_{15}$, $RD_1 = LE_{15}$

In general, for the i th iteration of the encryption algorithm,

$$\begin{aligned} LE_i &= RE_{i-1} \\ RE_i &= LE_{i-1} \oplus F(LE_{i-1}, K_i) \end{aligned}$$

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

2.11 BLOCK CIPHER MODES OF OPERATION

- Block Cipher is the basic building block to provide data security.
- To apply the block cipher to various applications, NIST has proposed 4 modes of operation. The block cipher is used to enhance the security of the encryption algorithm

2.11.1 Multiple Encryption and Triple DES

The vulnerability of DES to a brute-force attack has been detected by using two approaches are shown in figure 2.13

1. One approach is to design a completely new algorithm, of which AES is a prime example
2. Another alternative, which would preserve the existing investment in software and equipment, is to use multiple encryptions with DES and multiple keys.

Double DES

The simplest form of multiple encryptions has two encryption stages and two keys. Given a plaintext P and two encryption keys K_1 and K_2 , cipher text C is generated as

$$C = E(K_2, E(K_1, P))$$

Decryption requires that the keys be applied in reverse order:

$$P = D(K_1, D(K_2, C))$$

For DES, this scheme apparently involves a key length of $56 * 2 = 112$ bits, resulting in a dramatic increase in cryptographic strength.

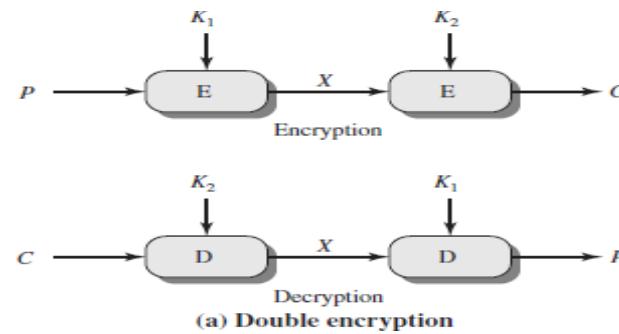


Figure 2.13 Multiple Encryption

Reduction to a Single Stage

Suppose it were true for DES, for all 56-bit key values, that given any two keys K_1 and K_2 , it would be possible to find a key K_3 such that

$$E(K_2, E(K_1, P)) = E(K_3, P)$$

Meet-in-the-Middle Attack

The use of double DES results in a mapping that is not equivalent to a single DES encryption. But there is a way to attack this scheme, one that does not depend on any particular property of DES but that will work against any block encryption cipher. This algorithm, known as a meet-in-the-middle attack.

It is based on the observation that, if we have

$$C = E(K_2, E(K_1, P))$$

Then

$$X = E(K_1, P) = D(K_2, C)$$

Given a known pair, (P, C) , the attack proceeds as follows. First, encrypt P for all 256 possible values of K_1 . Store these results in a table and then sort the table by the Values of X .

Next, decrypt C using all 256 possible values of K_2 . As each decryption is produced, check the result against the table for a match.

If a match occurs, then test the two resulting keys against a new known plaintext–cipher text pair. If the two keys produce the correct cipher text, accept them as the correct keys.

For any given plaintext P , there are 264 possible cipher text values that could be produced by double DES. Double DES uses, in effect, a 112-bit key, so that there are 2112 possible keys.

Triple DES with Two Keys

To overcome the meet-in-the-middle attack is to use three stages of encryption with three different keys. This is called ad Triple DES or 3DES as shown in figure 2.14.

The known plain text attack in 2^{112} . The key length of $56 * 3 = 168$ bits which is a drawback.

Tuchman proposed a triple encryption method that uses only two keys given plain text k_1, k_2 . The final cipher text is

$$C = E(K_1, D(K_2, E(K_1, P)))$$

$$P = D(K_1, E(K_2, D(K_1, C)))$$

- The function follows an encrypt-decrypt-encrypt (EDE) sequence

Its only advantage is that it allows users of 3DES to decrypt data encrypted by users of the older single DES:

$$C = E(K_1, D(K_1, E(K_1, P))) = E(K_1, P)$$

$$P = D(K_1, E(K_1, D(K_1, C))) = D(K_1, C)$$

- 3DES with two keys is a relatively popular alternative to DES
- There are no practical cryptanalytic attacks on 3DES.
- The cost of a brute-force key search on 3DES is on the order of 2^{112}

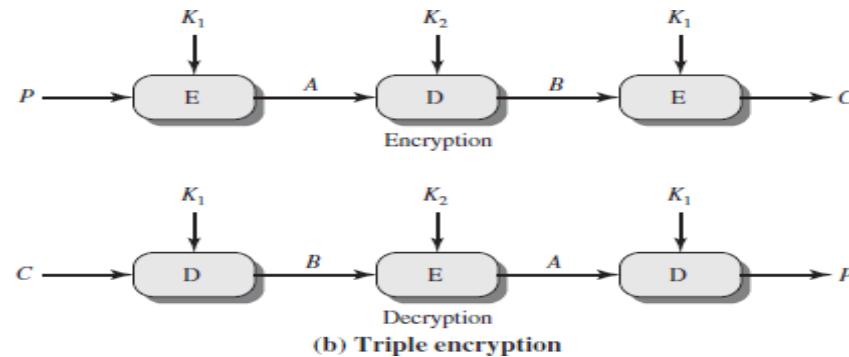


Figure 2.14 Triple DES

The first serious proposal came from Merkle and Hellman

1. Merkle and Hellman

The concept is to find plaintext values that produce a first intermediate value of $A = 0$ and then using the meet-in-the-middle attack to determine the two keys.

- The level of effort is 2^{56} ,
- The technique requires 256 chosen plaintext–cipher text pairs, which is a number unlikely to be provided.

2. known - plaintext attack:

The attack is based on the observation that if we know A and C then the problem reduces to that of an attack on double DES.

The attacker does not know A, even if P and C are known, as long as the two keys are unknown. The attacker can choose a potential value of A and then try to find a known (P, C) pair that produces A.

The attack proceeds as follows.

Step 1:

- Obtain n (P, C) pairs. This is the known plaintext. Place these in a table sorted on the values of P

Step 2:

- Pick an arbitrary value a for A, and create a second table with entries defined in the following fashion.
- For each of the 2^{56} possible keys $K_1 = i$, calculate the plaintext value P_i that produces a.
- For each P_i that matches an entry in Table 1, create an entry in Table 2 consisting of the K_1 value and the value of B that is produced.

Step 3:

- We now have a number of candidate values of K_1 in Table 2 and are in a position to search for a value of K_2 .
- For each of the 256 possible keys $K_2 = j$, calculate the second intermediate value for our chosen value of a .
- If there is a match, then the corresponding key i from Table 2 plus this value of j are candidate values for the unknown keys (K_1, K_2).

Step 4:

- Test each candidate pair of keys (i, j) on a few other plaintext–cipher text pairs.
- If a pair of keys produces the desired cipher text, the task is complete. If no pair succeeds, repeat from step 1 with a new value of a .

2.11.2 MODE 1: Electronic Code Book

The simplest mode is the electronic codebook (ECB) mode shown in figure 2.15. Here plaintext is handled one block at a time and each block of plaintext is encrypted using the same key.

The term codebook is used because, for a given key, there is a unique cipher text for every b -bit block of plaintext.

When the message longer than b bits, to break the message into b -bit blocks. For the last block when the no of bits is less than b , padding the last block if necessary.

Decryption is performed one block at a time, always using the same key.

Uses: The ECB method is ideal for a short amount of data, such as an encryption key.

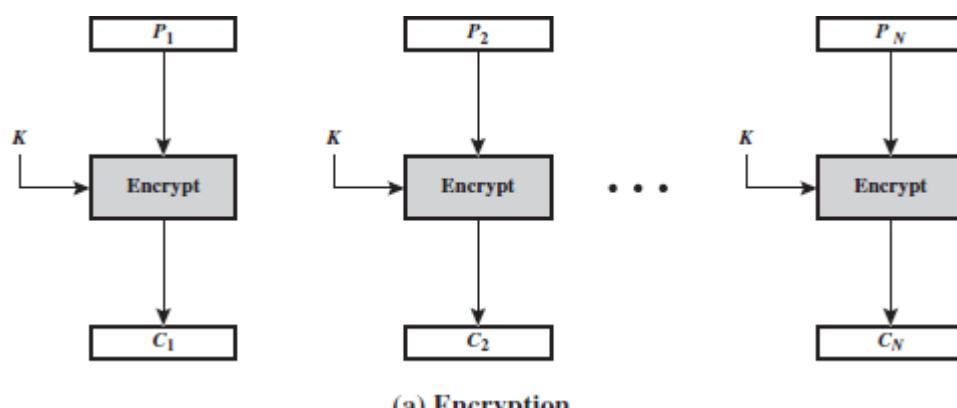
Disadvantage:

When b -bit block of plaintext appears more than once in the message, it always produces the same cipher text output.

For lengthy messages, the ECB mode may not be secure. If the message is highly structured, it may be possible for a cryptanalyst to exploit these regularities.

If the message has repetitive elements with a period of repetition a multiple of b bits, then these elements can be identified by the analyst.

This may help in the analysis or may provide an opportunity for substituting or rearranging blocks.



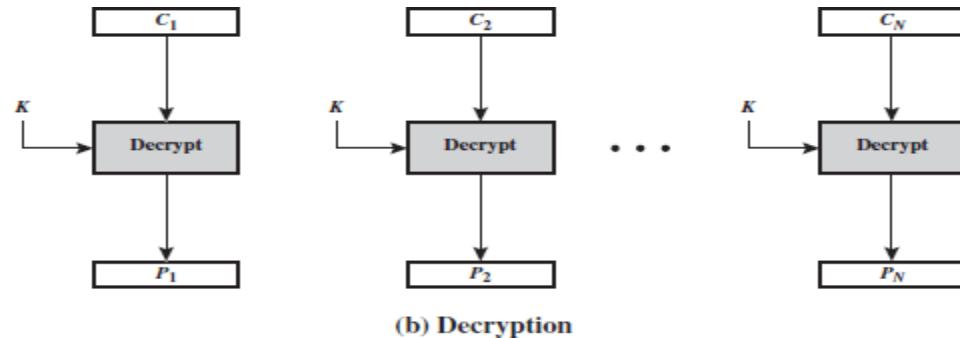


Figure 2.15 Electronic Code Book (ECB)

Mode Properties for Evaluating and Constructing ECB

Overhead: The additional operations for the encryption and decryption operation when compared to encrypting and decrypting in the ECB mode.

Error recovery: The property that an error in the i th cipher text block is inherited by only a few plaintext blocks

Error propagation: It is meant here is a bit error that occurs in the transmission of a cipher text block, not a computational error in the encryption of a plaintext block.

Diffusion: Low entropy plaintext blocks should not be reflected in the cipher text blocks. Roughly, low entropy equates to predictability or lack of randomness

Security: Whether or not the cipher text blocks leak information about the plaintext blocks.

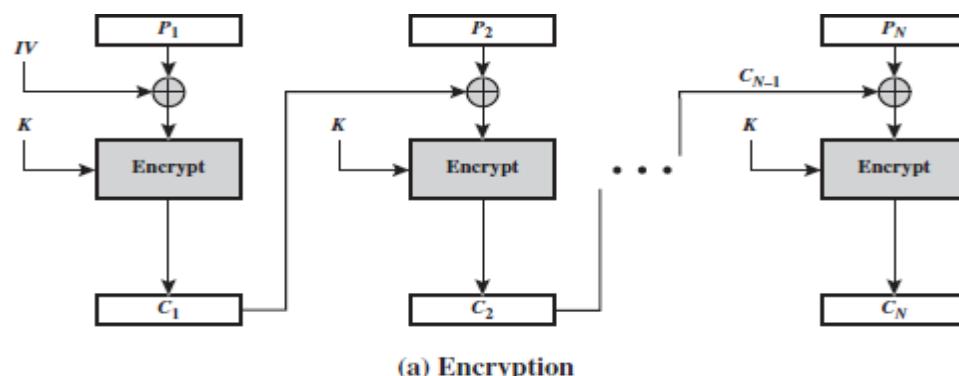
2.11.3 MODE 2: Cipher Block Chaining Mode

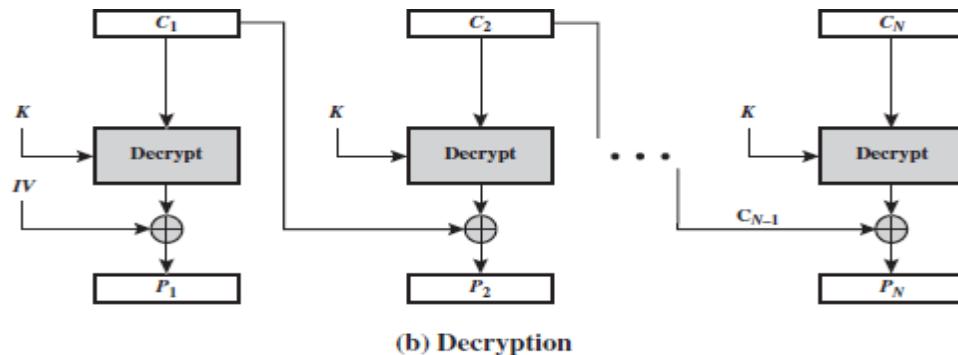
This method is to overcome the disadvantage of ECB (i.e) when the PT block is repeated CBC produces different cipher text blocks

The input to the encryption function for each plaintext block bears no fixed relationship to the plaintext block. Therefore, repeating patterns of b bits are not exposed.

For decryption, each cipher block is passed through the decryption algorithm. The result is XORed with the preceding cipher text block to produce the plaintext block are shown in figure 2.16.

$$C_j = E(K, [C_{j-1} \oplus P_j])$$



**Figure 2.16 Cipher Block Chaining (CBC) Mode**

Then

$$D(K, C_j) = D(K, E(K, [C_{j-1} \oplus P_j]))$$

To produce the first block of cipher text, an initialization vector (IV) is XORed with the first block of plaintext.

On decryption, the IV is XORed with the output of the decryption algorithm to recover the first block of plaintext.

Size of IV = Size of data Blocks

We can define CBC mode as

CBC	$C_1 = E(K, [P_1 \oplus IV])$ $C_j = E(K, [P_j \oplus C_{j-1}]) \quad j = 2, \dots, N$	$P_1 = D(K, C_1) \oplus IV$ $P_j = D(K, C_j) \oplus C_{j-1} \quad j = 2, \dots, N$
-----	-------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------

For maximum security, the IV should be protected against unauthorized changes. This could be done by sending the IV using ECB encryption

Reason for protecting the IV:

If an opponent is able to fool the receiver into using a different value for IV, then the opponent is able to invert selected bits in the first block of plaintext. To see this, consider

$$\begin{aligned} C_1 &= E(K, [IV \oplus P_1]) \\ P_1 &= IV \oplus D(K, C_1) \end{aligned}$$

Now use the notation that $X[i]$ denotes the i th bit of the b -bit quantity X . Then

$$P_1[i] = IV[i] \oplus D(K, C_1)[i]$$

Then, using the properties of XOR, we can state

$$P_1[i]' = IV[i]' \oplus D(K, C_1)[i]$$

Where the prime notation denotes bit complementation. This means that if an opponent can predictably change bits in IV, the corresponding bits of the received value of P_1 can be changed.

2.11.4 MODE 3: Cipher Feedback Mode:

We know that the DES is a block cipher. It is possible to convert block cipher into stream Cipher using CFB mode

The advantages of CFB is that

- Eliminates the need to pad a message
- It also can operate in real time
- The length of the CT = Length of PT

Figure 2.17 depicts the CFB scheme. In the figure 2.17, it is assumed that the unit of transmission is s bits; a common value is s = 8.

The units of plaintext are chained together; to get the cipher text is a function of all preceding plaintext. Here the plaintext is divided into segments of s bits.

Encryption:

The input to the encryption function is a b-bit shift register that is initially set to some initialization vector (IV).

The leftmost (most significant) s bits of the output of the encryption function are XORed with the first segment of plaintext P1 to produce the first unit of cipher text C1.

The contents of the shift register are shifted left by s bits, and C1 is placed in the rightmost (least significant) s bits of the shift register.

This process continues until all plaintext units have been encrypted.

Decryption:

The same scheme is used, except that the received cipher text unit is XORed with the output of the encryption function to produce the plaintext unit.

Let MSBs(X) be defined as the most significant s bits of X. Then

$$C_1 = P_1 \oplus \text{MSB}_s[\text{E}(K, \text{IV})]$$

Therefore, by rearranging terms:

$$P_1 = C_1 \oplus \text{MSB}_s[\text{E}(K, \text{IV})]$$

The same reasoning holds for subsequent steps in the process.

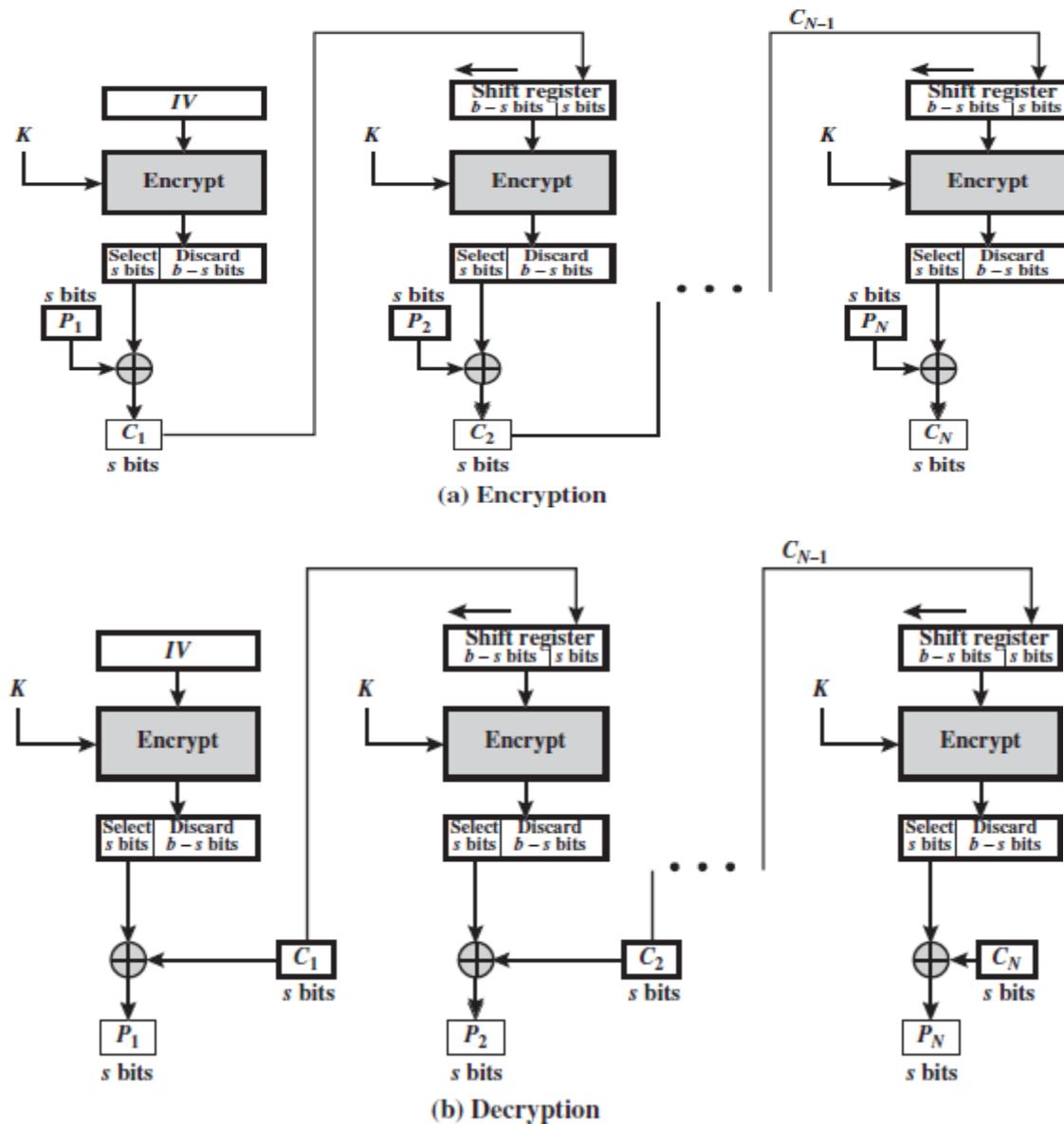


Figure 2.17 S-bit Cipher Feedback (CFB) mode

We can define CFB mode as follows

CFB	$I_1 = IV$	$I_1 = IV$
	$I_j = \text{LSB}_{b-s}(I_{j-1}) \ C_{j-1} \quad j = 2, \dots, N$	$I_j = \text{LSB}_{b-s}(I_{j-1}) \ C_{j-1} \quad j = 2, \dots, N$
	$O_j = E(K, I_j) \quad j = 1, \dots, N$	$O_j = E(K, I_j) \quad j = 1, \dots, N$
	$C_j = P_j \oplus \text{MSB}_s(O_j) \quad j = 1, \dots, N$	$P_j = C_j \oplus \text{MSB}_s(O_j) \quad j = 1, \dots, N$

2.11.5 Output Feedback Mode

The output feedback (OFB) mode is similar in structure to that of CFB.

The output of the encryption function is fed back to become the input for encrypting the next block of plaintext as shown in figure 2.18.

Comparison between OFB and CFB

In CFB, the output of the XOR unit is fed back to become input for encrypting the next block.

The other difference is that the OFB mode operates on full blocks of plaintext and cipher text, whereas CFB operates on an s-bit subset. OFB encryption can be expressed as

Where

$$C_j = P_j \oplus E(K, O_{j-1})$$

$$O_{j-1} = E(K, O_{j-2})$$

we can rewrite the encryption expression as:

$$C_j = P_j \oplus E(K, [C_{j-1} \oplus P_{j-1}])$$

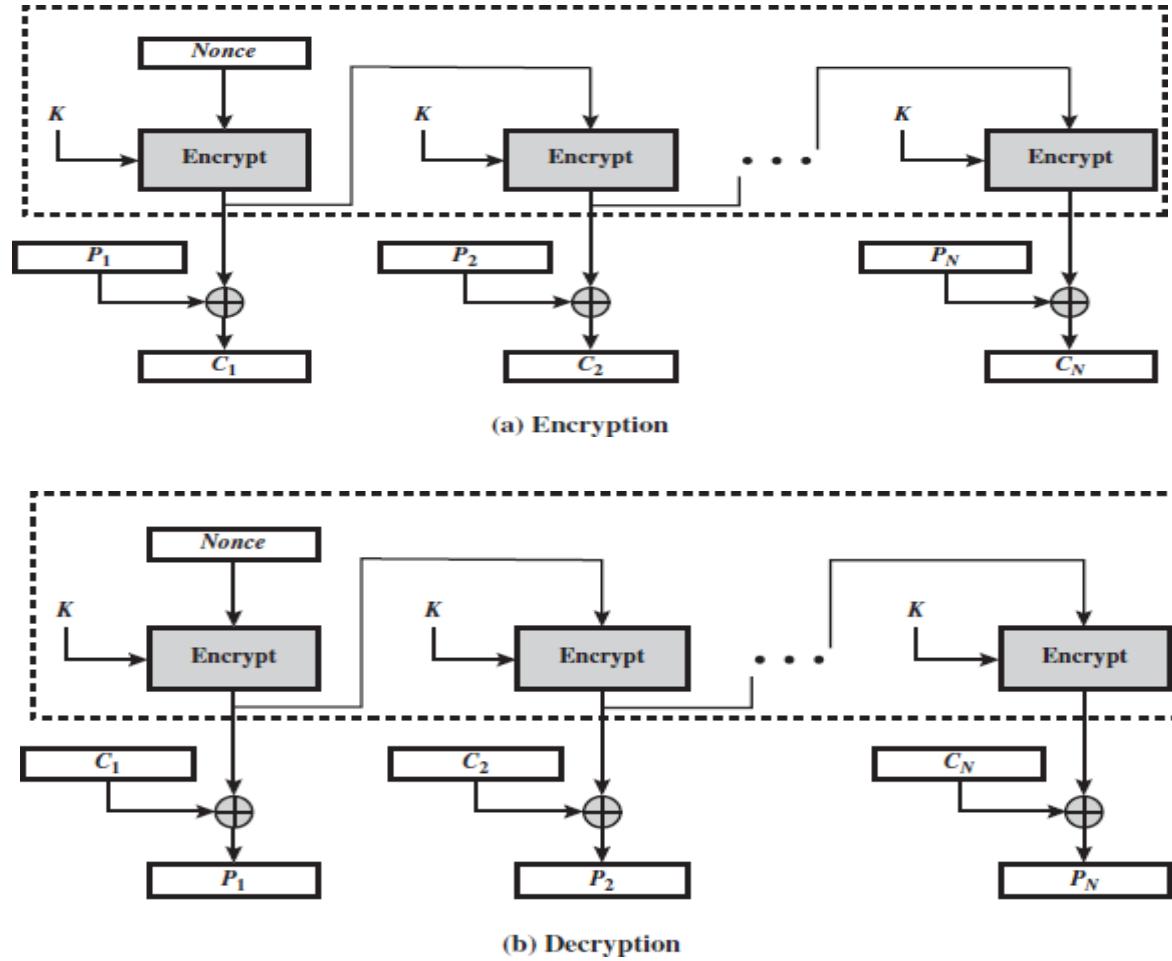
By rearranging terms, we can demonstrate that decryption works.

$$P_j = C_j \oplus E(K, [C_{j-1} \oplus P_{j-1}])$$

We can define OFB mode as follows.

OFB	$I_1 = \text{Nonce}$ $I_j = O_{j-1} \quad j = 2, \dots, N$ $O_j = E(K, I_j) \quad j = 1, \dots, N$ $C_j = P_j \oplus O_j \quad j = 1, \dots, N - 1$ $C_N^* = P_N^* \oplus \text{MSB}_u(O_N)$	$I_1 = \text{Nonce}$ $I_j = O_{j-1} \quad j = 2, \dots, N$ $O_j = E(K, I_j) \quad j = 1, \dots, N$ $P_j = C_j \oplus O_j \quad j = 1, \dots, N - 1$ $P_N^* = C_N^* \oplus \text{MSB}_u(O_N)$
------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Let the size of a block be b. If the last block of plaintext contains u bits (indicated by *), with $u < b$, the most significant u bits of the last output block O_N are used for the XOR operation. The remaining $b - u$ bits of the last output block are discarded.

**Figure 2.18 Output Feedback Mode****Advantage:**

Bit errors in transmission do not propagate (i.e.) when bit errors occurs in C_i , P_i is alone affected

Disadvantage:

Vulnerable to message stream modification attack

2.11.6 Counter Mode

The counter (CTR) mode has increased recently with applications to ATM (asynchronous transfer mode) network security and IP sec (IP security).

A counter equal to the plaintext block size is used. The counter value must be different for each plaintext block as shown in figure 2.19.

The counter is initialized to some value and then incremented by 1 for each subsequent block (modulo 2^b , where b is the block size). For encryption, the counter is encrypted and then XORed with the plaintext block to produce the cipher text block.

For decryption, the same sequence of counter values is used, with each encrypted counter XORed with a cipher text block to recover the corresponding plaintext block.

Advantage:

Hardware efficiency

- CTR can be done in parallel

Software efficiency

- CTR supports parallel feature pipelining

Preprocessing

Simplicity

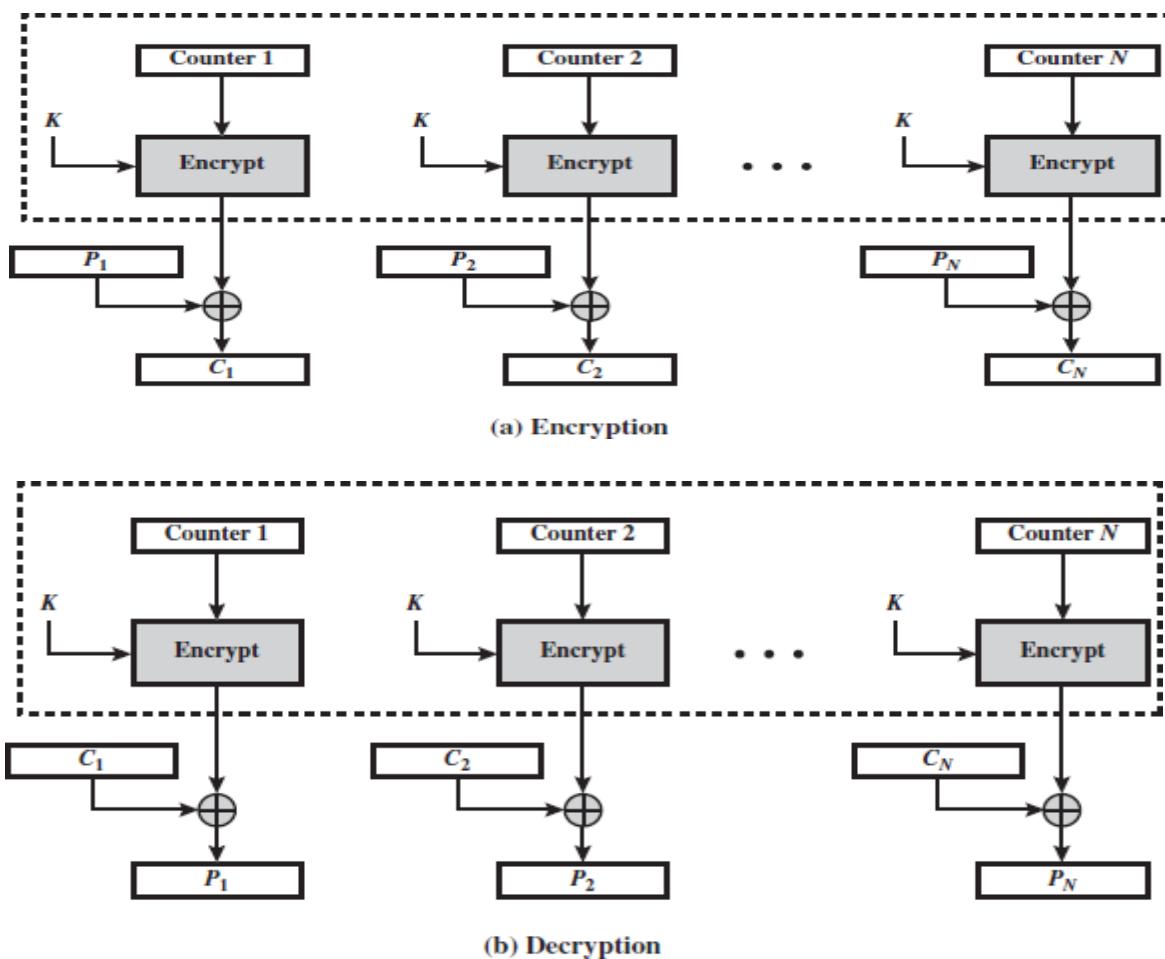


Figure 2.19 Counter Mode

2.12 ADVANCED ENCRYPTION STANDARD (AES)

AES is a symmetric block cipher that is intended to replace DES as the approved standard for a wide range of applications. Compared to public-key ciphers such as RSA, the structure of AES and most symmetric ciphers is quite complex and cannot be explained as easily as many other cryptographic algorithms.

2.12.1 Finite Field Arithmetic

In AES, all operations are performed on 8-bit bytes. The arithmetic operations of addition, multiplication, and division are performed over the finite field GF. A field is a set in which we can do addition, subtraction, multiplication, and division without leaving the set. Division is defined with the following rule: $a/b = a(b-1)$.

An example of a finite field (one with a finite number of elements) is the set Z_p consisting of all the integers $\{0, 1, 2, \dots, p-1\}$, where p is a prime number and in which arithmetic is carried out modulo p .

The way of defining a finite field containing 2^n elements; such a field is referred to as $GF(2^n)$. Consider the set, S , of all polynomials of degree $n - 1$ or less with binary coefficients. Thus, each polynomial has the form

$$f(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \dots + a_1x + a_0 = \sum_{i=0}^{n-1} a_i x^i$$

Where each a_i takes on the value 0 or 1. There are a total of 2^n different polynomials in S . For $n = 3$, the $2^3 = 8$ polynomials in the set are

$$\begin{array}{cccc} 0 & x & x^2 & x^2 + x \\ 1 & x + 1 & x^2 + 1 & x^2 + x + 1 \end{array}$$

Appropriate definition of arithmetic operations, each such set S is a finite field.

The definition consists of the following elements.

1. Arithmetic follows the ordinary rules of polynomial arithmetic using the basic rules of algebra with the following two refinements.
2. Arithmetic on the coefficients is performed modulo 2. This is the same as the XOR operation.
3. If multiplication results in a polynomial of degree greater than $n - 1$, then the polynomial is reduced modulo some irreducible polynomial $m(x)$ of degree n . That is, we divide by $m(x)$ and keep the remainder. For a polynomial $f(x)$, the remainder is expressed as $r(x) = f(x) \bmod m(x)$. A polynomial $m(x)$ is called **irreducible** if and only if $m(x)$ cannot be expressed as a product of two polynomials, both of degree lower than that of $m(x)$.

A polynomial in $GF(2^n)$ can be uniquely represented by its n binary coefficients ($a_{n-1}a_{n-2}\dots a_0$). Therefore, every polynomial in $GF(2^n)$ can be represented by an n -bit number.

2.12.2 AES Structure

General Structure

- Figure 2.20 shows the overall structure of the AES encryption process. The cipher takes a plaintext block size of 128 bits, or 16 bytes. The key length can be 16, 24, or 32 bytes (128, 192, or 256 bits). The algorithm is referred to as AES-128, AES-192, or AES-256, depending on the key length.

- The input to the encryption and decryption algorithms is a single 128-bit block. The block is depicted as a 4×4 square matrix of bytes. This block is copied into the **State** array, which is modified at each stage of encryption or decryption. After the final stage, **State** is copied to an output matrix. These operations are depicted in Figure 2.21a. Similarly, the key is depicted as a square matrix of bytes. This key is then expanded into an array of key schedule words.
- Below Figure 2.20 shows the expansion for the 128-bit key. Each word is four bytes, and the total key schedule is 44 words for the 128-bit key. Note that the ordering of bytes within a matrix is by column. The first four bytes of a 128-bit plaintext input to the encryption cipher occupy the first column of the **in** matrix. The second four bytes occupy the second column, and so on. Similarly, the first four bytes of the expanded key, which form a word, occupy the first column of the **w** matrix. The cipher consists of N rounds, where the number of rounds depends on the key length: 10 rounds for a 16-byte key, 12 rounds for a 24-byte key, and 14 rounds for a 32-byte key (Table 2.3).
- The first $N - 1$ round consist of four distinct transformation functions: Sub Bytes, Shift Rows, Mix Columns, and AddRoundKey, which are described subsequently. The final round contains only three transformations, and there is an initial single transformation (AddRoundKey) before the first round, which can be considered Round 0. Each transformation takes one or more 4×4 matrices as input and produces a 4×4 matrix as output. Figure 5.1 shows that the output of each round is a 4×4 matrix, with the output of the final round being the cipher text.

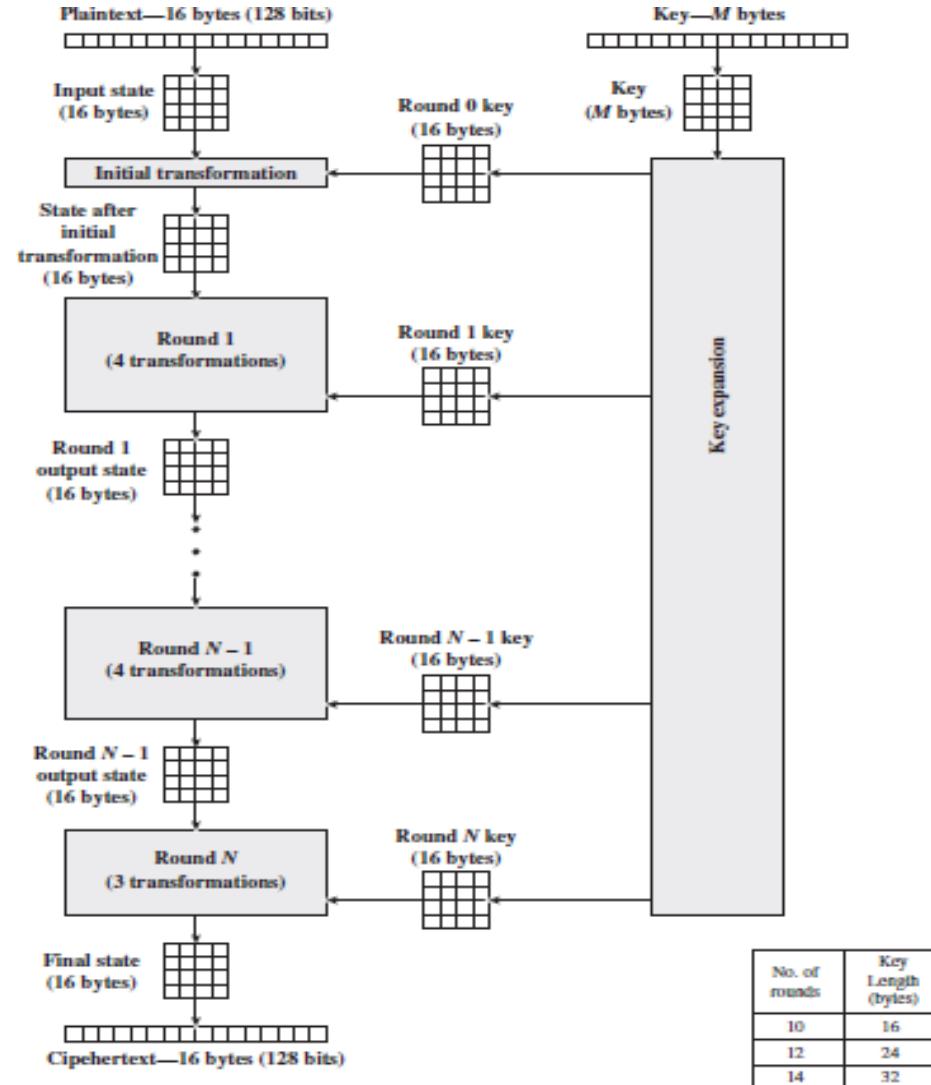


Figure 2.20 AES Encryption Process

Key Size (words/bytes/bits)	4/16/128	6/24/192	8/32/256
Plaintext Block Size (words/bytes/bits)	4/16/128	4/16/128	4/16/128
Number of Rounds	10	12	14
Round Key Size (words/bytes/bits)	4/16/128	4/16/128	4/16/128
Expanded Key Size (words/bytes)	44/176	52/208	60/240

Table 2.3 AES Parameters

2.12.3 Detailed Structure

Below Figure 2.20 shows the AES cipher shows the sequence of transformations in each round and showing the corresponding decryption function.

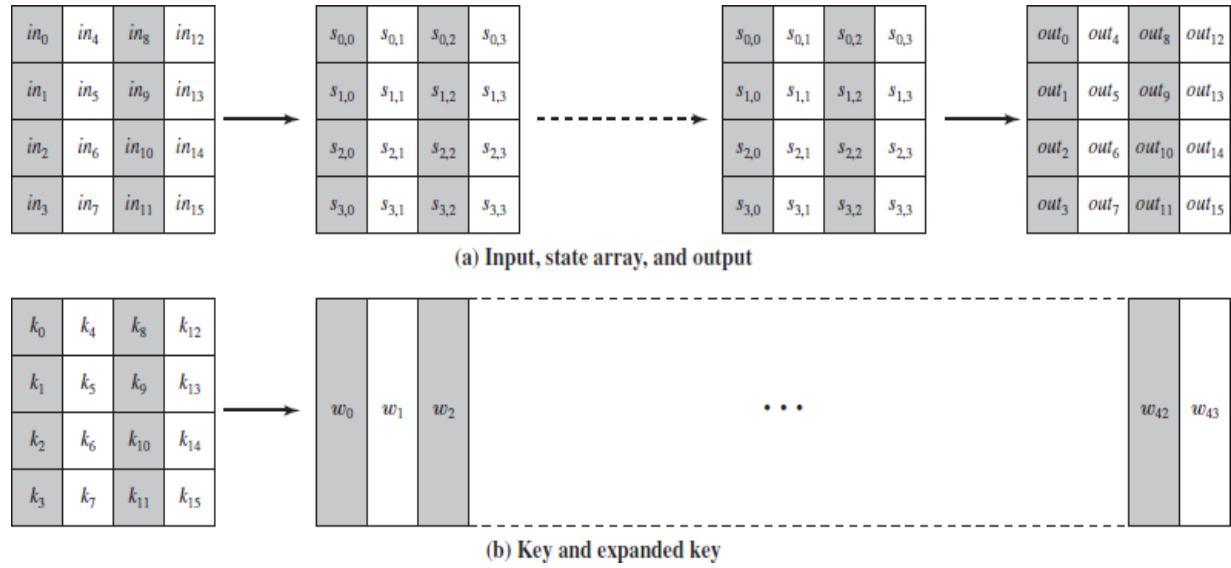


Fig: 2.21 Detail AES structure

Overall detail about AES structure.

1. It is not a Feistel structure. Recall that, in the classic Feistel structure, half of the data block is used to modify the other half of the data block and then the halves are swapped. AES instead processes the entire data block as a single matrix during each round using substitutions and permutation.
2. The key that is provided as input is expanded into an array of forty-four 32-bitwords, $w[j]$. Four distinct words (128 bits) serve as a round key for each round as shown in figure 2.22;
3. Four different stages are used, one of permutation and three of substitution:
 - **Substitute bytes:** Uses an S-box to perform a byte-by-byte substitution of the block
 - **ShiftRows:** A simple permutation
 - **MixColumns:** A substitution that makes use of arithmetic over GF(2⁸)
 - **AddRoundKey:** A simple bitwise XOR of the current block with a portion of the expanded key
4. The structure is quite simple. For both encryption and decryption as shown in figure 2.22, the cipher begins with an AddRoundKey stage, followed by nine rounds that each includes all four stages, followed by a tenth round of three stages.
5. Only the AddRoundKey stage makes use of the key. The AddRoundKey stage would provide no security because they do not use the key. We can view the cipher as alternating operations of XOR encryption (AddRoundKey) of a block, followed by scrambling of the block (the other three stages), followed by XOR encryption, and so on. This scheme is both efficient and highly secure.

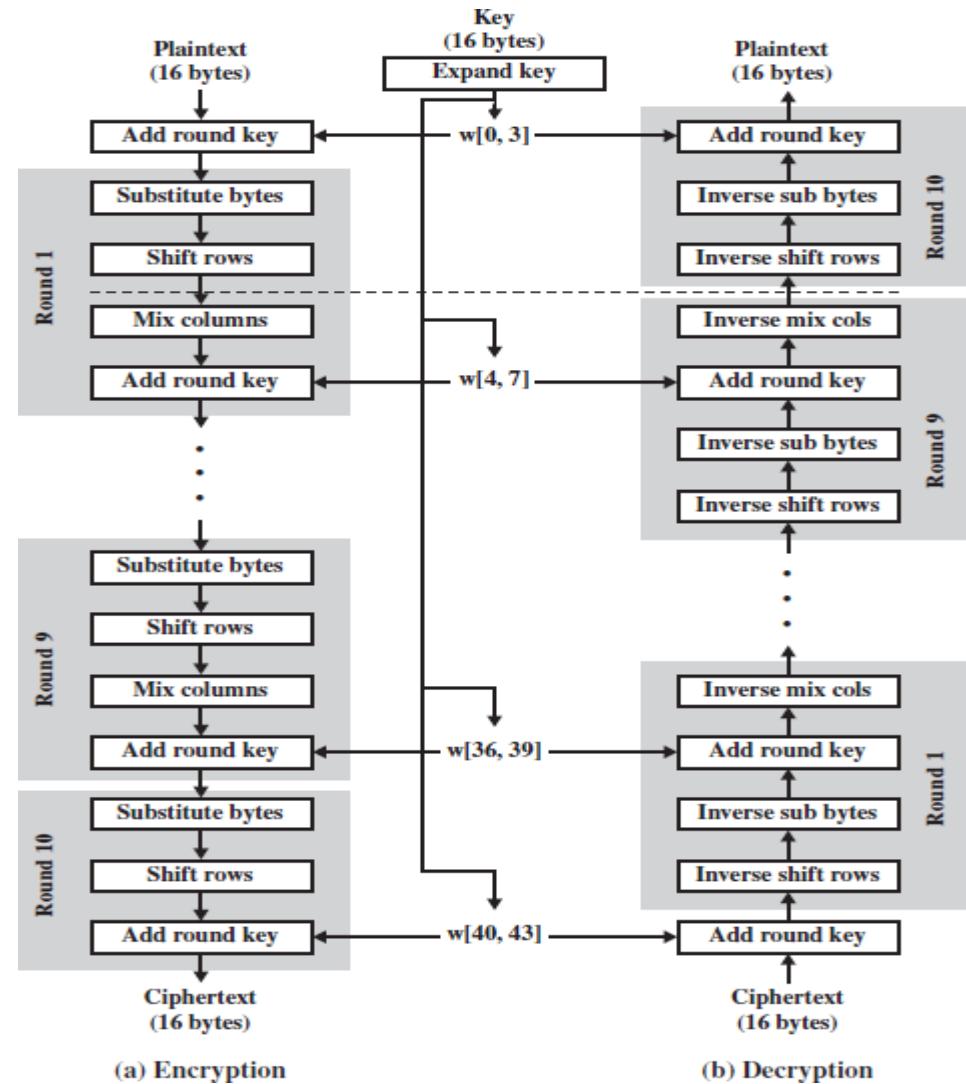
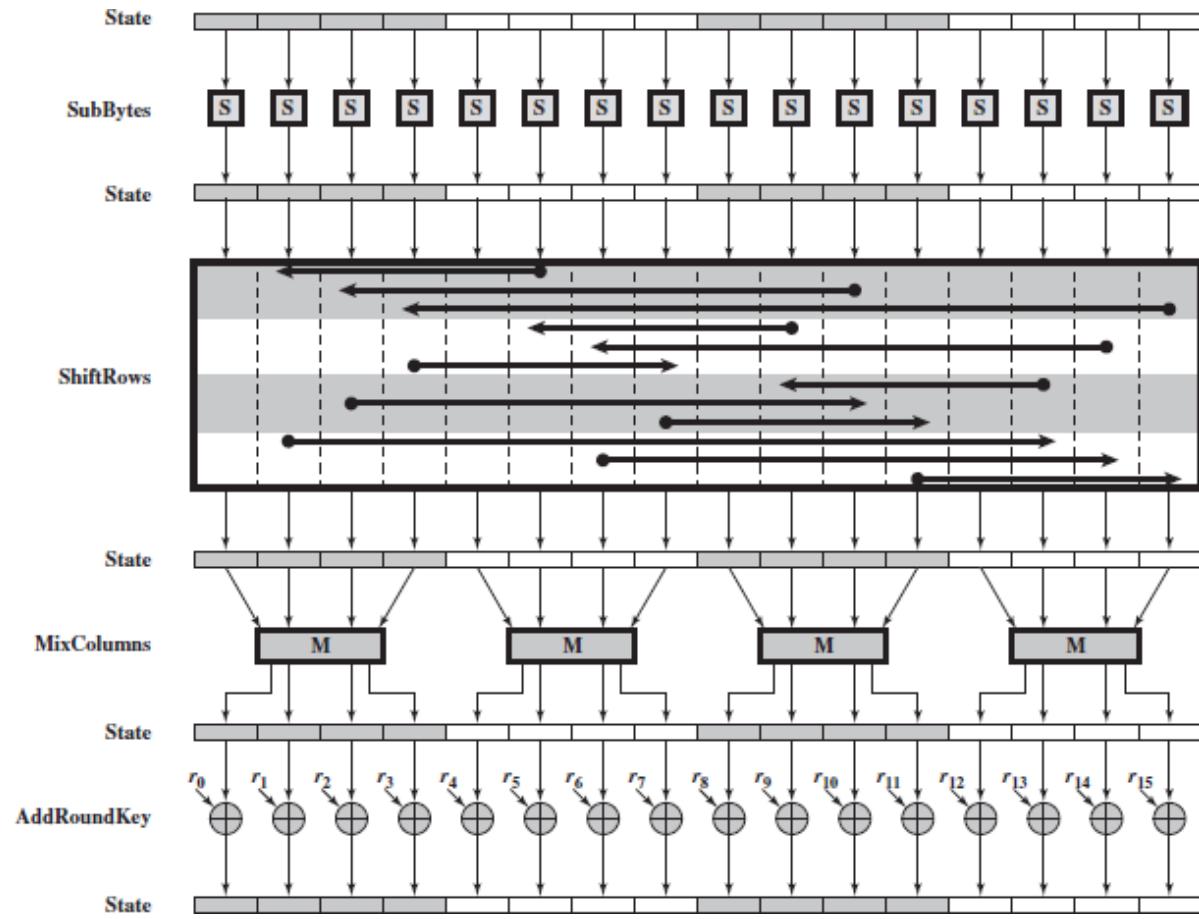


Fig 2.22 AES Encryption and Decryption

6. Each stage is easily reversible. For the Substitute Byte, ShiftRows, and MixColumns stages, an inverse function is used in the decryption algorithm. For the AddRoundKey stage, the inverse is achieved by XORing the same round key to the block, using the result that.

$$A \oplus B \oplus B = A$$

7. The decryption algorithm makes use of the expanded key in reverse order. However, the decryption algorithm is not identical to the encryption algorithm. This is a consequence of the particular structure of AES.

**Fig 2.23 AES Encryption Round**

8. Once it is established that all four stages are reversible, it is easy to verify that decryption does recover the plaintext.
9. The final round of both encryption and decryption consists of only three stages. Again, this is a consequence of the particular structure of AES and is required, to make the cipher reversible

2.12.4 AES Transformation Functions

Four transformations used in AES. For each stage, we describe the forward (encryption) algorithm, the inverse (decryption) algorithm, and the rationale for the stage.

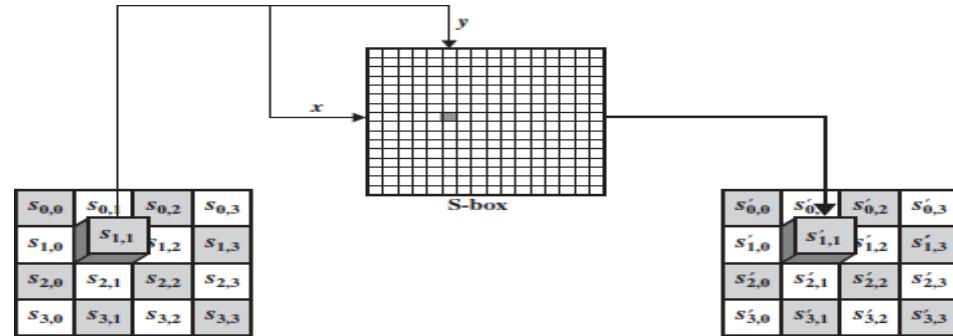
Substitute Bytes Transformation

Type 1: Forward and Inverse Transformations:

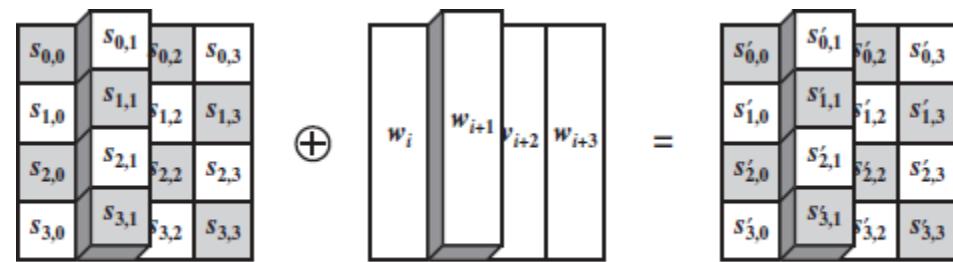
The forward substitute byte transformation, called Sub Bytes, is a simple table lookup (Figure 2.24a). AES defines a 16×16 matrix of byte values, called an S-box that contains a permutation of all possible 256 8-bit values.

Each individual byte of **State** is mapped into a new byte in the following way: The leftmost 4 bits of the byte are used as a row value and the rightmost 4 bits are used as a column value. These row and column values serve as indexes into the S-box to select a unique 8-bit output value as shown in figure 2.25.

For example, the hexadecimal value {95} references row 9, column 5 of the S-box, which contains the value {2A}. Accordingly, the value {95} is mapped into the value {2A}.



(a) Substitute byte transformation



(b) Add round key transformation

Figure 2.24 AES Byte level Operations

		y															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
x	0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
	1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
	2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
	3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
	4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
	5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
	6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
	7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
	8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
	9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
	A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
	B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
	C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
	D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
	E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
	F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

(a) S-box

		y															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
x	0	52	09	6A	D5	30	36	A5	38	BF	40	A3	9E	81	F3	D7	FB
	1	7C	E3	39	82	9B	2F	FF	87	34	8E	43	44	C4	DE	E9	CB
	2	54	7B	94	32	A6	C2	23	3D	EE	4C	95	0B	42	FA	C3	4E
	3	08	2E	A1	66	28	D9	24	B2	76	5B	A2	49	6D	8B	D1	25
	4	72	F8	F6	64	86	68	98	16	D4	A4	5C	CC	5D	65	B6	92
	5	6C	70	48	50	FD	ED	B9	DA	5E	15	46	57	A7	8D	9D	84
	6	90	D8	AB	00	8C	BC	D3	0A	F7	E4	58	05	B8	B3	45	06
	7	D0	2C	1E	8F	CA	3F	0F	02	C1	AF	BD	03	01	13	8A	6B
	8	3A	91	11	41	4F	67	DC	EA	97	F2	CF	CE	F0	B4	E6	73
	9	96	AC	74	22	E7	AD	35	85	E2	F9	37	E8	1C	75	DF	6E
	A	47	F1	1A	71	1D	29	C5	89	6F	B7	62	0E	AA	18	BE	1B
	B	FC	56	3E	4B	C6	D2	79	20	9A	DB	C0	FE	78	CD	5A	F4
	C	1F	DD	A8	33	88	07	C7	31	B1	12	10	59	27	80	EC	5F
	D	60	51	7F	A9	19	B5	4A	0D	2D	E5	7A	9F	93	C9	9C	EF
	E	A0	E0	3B	4D	AE	2A	F5	B0	C8	EB	BB	3C	83	53	99	61
	F	17	2B	04	7E	BA	77	D6	26	E1	69	14	63	55	21	0C	7D

(b) Inverse S-box

Figure 2.25 AES S-Boxes

Here is an example of the SubBytes transformation:

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

→

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

The S-box is constructed in the following fashion (Figure 2.26a).

1. Initialize the S-box with the byte values in ascending sequence row by row. The first row contains {00}, {01}, {02}, ..., {0F}; the second row contains {10}, {11}, etc.; and so on. Thus, the value of the byte at row y , column x is $\{yx\}$.
2. Map each byte in the S-box to its multiplicative inverse in the finite field GF(28); the value {00} is mapped to itself.
3. Consider that each byte in the S-box consists of 8 bits labeled $(b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0)$. Apply the following transformation to each bit of each byte in the S-box:

$$b'_i = b_i \oplus b_{(i+4) \bmod 8} \oplus b_{(i+5) \bmod 8} \oplus b_{(i+6) \bmod 8} \oplus b_{(i+7) \bmod 8} \oplus c_i$$

Where c_i is the i th bit of byte c with the value {63}; that is, $(c_7 c_6 c_5 c_4 c_3 c_2 c_1 c_0) = (01100011)$. The prime („) indicates that the variable is to be updated by the value on the right.

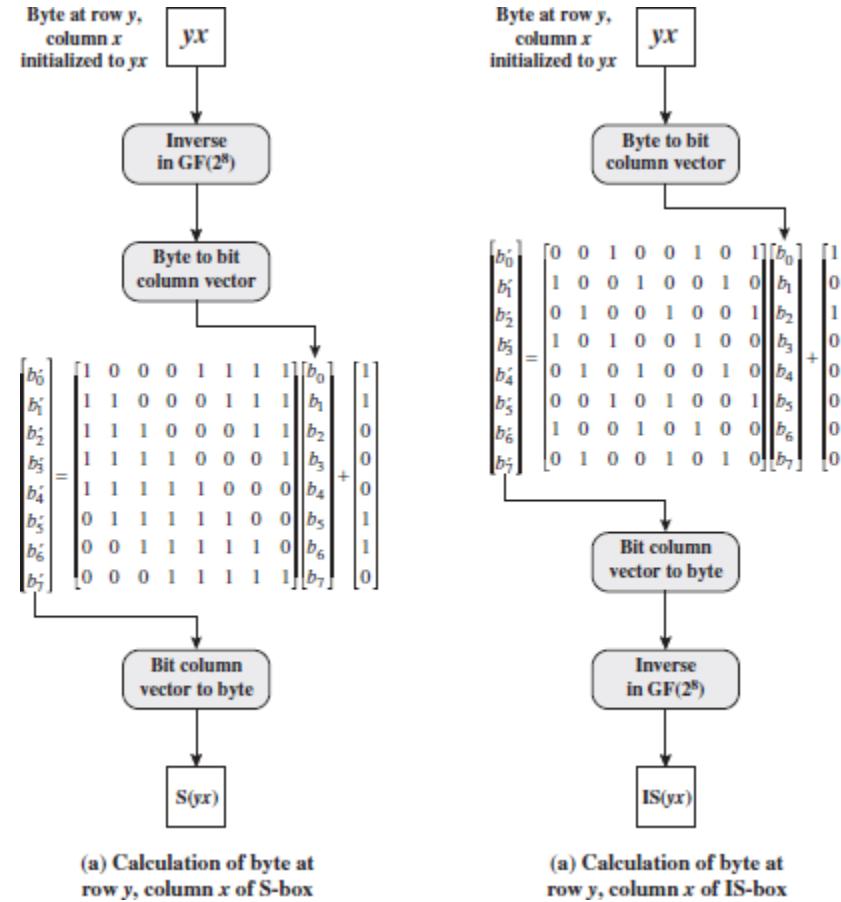


Figure 2.26 Construction of S-Box and IS-Box

$$\begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

The AES standard depicts this transformation in matrix form as follows.

- In ordinary matrix multiplication, each element in the product matrix is the sum of products of the elements of one row and one column. Each element in the product matrix is the bitwise XOR of products of elements of one row and one column.
- As an example, consider the input value {95}. The multiplicative inverse in $GF(2^8)$ is $\{95\}^{-1} = \{8A\}$, which is 10001010 in binary. Using above Equation

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

The result is {2A}, which should appear in row {09} column {05} of the S-box.

Type 2: Inverse Substitute Byte Transformation:

The **inverse substitute byte transformation**, called InvSubBytes, For example, that the input {2A} produces the output {95}, and the input {95} to the S-box produces {2A}. The inverse S-box is constructed by applying the inverse of the transformation is followed by taking the

$$b'_i = b_{(i+2) \bmod 8} \oplus b_{(i+5) \bmod 8} \oplus b_{(i+7) \bmod 8} \oplus d_i$$

multiplicative inverse in GF(28). The inverse transformation is

where byte $d = \{05\}$, or 00000101. We can depict this transformation as follows.

$$\begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

InvSubBytes is the inverse of Sub Bytes, label the matrices in sub Bytes and InvSubBytes as **X** and **Y**, respectively, and the vector versions of constants c and d as **C** and **D**, respectively. For some 8-bit vector **B**, becomes $\mathbf{B}' = \mathbf{XB} \oplus \mathbf{C}$. We need to show that

Y(XB ⊕ C) ⊕ D = B. To multiply out, we must show $\mathbf{YXB} \oplus \mathbf{YC} \oplus \mathbf{D} = \mathbf{B}$. This becomes

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

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

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

We have demonstrated that \mathbf{YX} equals the identity matrix, and the $\mathbf{YC} = \mathbf{D}$, so that $\mathbf{YC} \oplus \mathbf{D}$ equals the null vector.

Type 3: Shift Rows Transformation

Forward and Inverse Shift Rows Transformations:

The **forward shift row transformation**, called Shift Rows, is depicted in Figure 2.27. The first row of **State** is not altered. For the second row, a 1-byte circular left shift is performed. For the third row, a 2-byte circular left shift is performed. For the fourth row, a 3-byte circular left shift is performed. The following is an example of Shift Rows

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

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

Figure 2.27 Forward Shift Row Transformation

The **inverse shift row transformation**, called InvShiftRows, performs the circular shifts in the opposite direction for each of the last three rows, with a 1-byte circular right shift for the second row, and as shown in figure 2.28

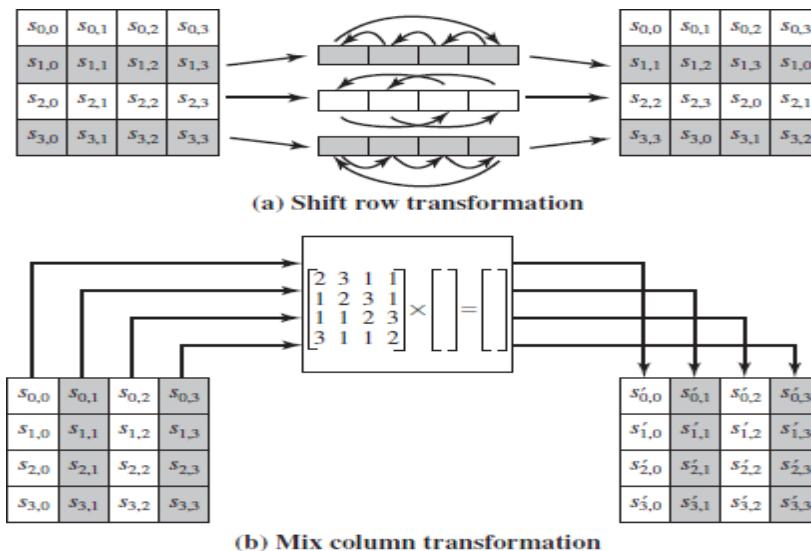


Figure 2.28 AES Row and Column Operations

Type 4: Mix Columns Transformation

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

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

Each element in the product matrix is the sum of products of elements of one row and one column. In this case, the individual additions and multiplications are performed in GF(2⁸).

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

The MixColumns transformation on a single column of State can be expressed as

The following is an example of MixColumns:

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

→

47	40	A3	4C
37	D4	70	9F
94	E4	3A	42
ED	A5	A6	BC

The MixColumns transformation on the first column, we need to show that

$$\begin{aligned} (\{02\} \cdot \{87\}) \oplus (\{03\} \cdot \{6E\}) \oplus \{46\} \oplus \{A6\} &= \{47\} \\ \{87\} \oplus (\{02\} \cdot \{6E\}) \oplus (\{03\} \cdot \{46\}) \oplus \{A6\} &= \{37\} \\ \{87\} \oplus \{6E\} \oplus (\{02\} \cdot \{46\}) \oplus (\{03\} \cdot \{A6\}) &= \{94\} \\ (\{03\} \cdot \{87\}) \oplus \{6E\} \oplus \{46\} \oplus (\{02\} \cdot \{A6\}) &= \{ED\} \end{aligned}$$

For the first equation, we have $\{02\} \cdot \{87\} = (0000\ 1110) \oplus (0001\ 1011) = (0001\ 0101)$ and

$$\{03\} \cdot \{6E\} = \{6E\} \oplus (\{02\} \cdot \{6E\}) = (0110\ 1110) \oplus (1101\ 1100) = (1011\ 0010) \text{ then}$$

$$\begin{aligned} \{02\} \cdot \{87\} &= 0001\ 0101 \\ \{03\} \cdot \{6E\} &= 1011\ 0010 \\ \{46\} &= 0100\ 0110 \\ \{A6\} &= \underline{1010\ 0110} \\ 0100\ 0111 &= \{47\} \end{aligned}$$

The **inverse mix column transformation**, called InvMixColumns, is defined by the following matrix multiplication:

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

The **inverse** of Equation need to show

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix}$$

That is, the inverse transformation matrix times the forward transformation matrix equals the identity matrix. To verify the first column of above Equation.

For the first equation, we have $\{0E\} \cdot \{02\} = 00011100$ and $\{09\} \cdot \{03\} = \{09\} \oplus \{09\} \cdot \{02\} = 00001001 \oplus 00010010 = 00011011$ then

$$\begin{aligned} \{0E\} \cdot \{02\} &= 00011100 \\ \{0B\} &= 00001011 \\ \{0D\} &= 00001101 \\ \{09\} \cdot \{03\} &= \underline{\underline{00011011}} \\ &00000001 \end{aligned}$$

The encryption was deemed more important than decryption for two reasons:

1. For the CFB and OFB cipher modes only encryption is used.
2. AES can be used to construct a message authentication code and for this, only encryption is used.

Type 5: AddRoundKey Transformation

Forward and Inverse Transformations

In the **forward add round key transformation**, called AddRoundKey, the 128 bits of **State** are bitwise XORed with the 128bits of the round key.

The operation is viewed as a column wise operation between the 4 bytes of a **State** column and one word of the roundkey; it can also be viewed as a byte-level operation.

The following is an example of AddRoundKey:

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

The first matrix is **State**, and the second matrix is the round key.

The **inverse add round key transformation** is identical to the forward addround key transformation, because the XOR operation is its own inverse.

The Figure 2.29 is another view of a single round of AES, emphasizing the mechanisms and inputs of each transformation.

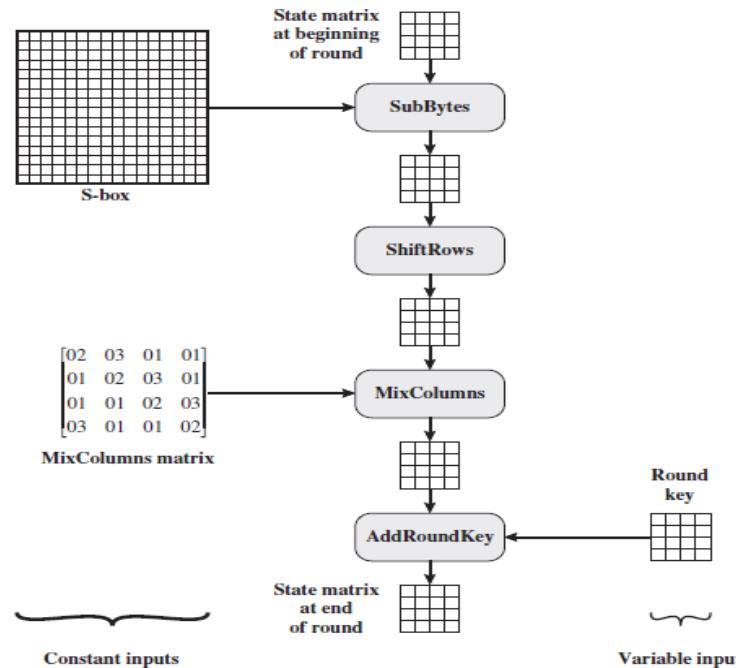


Fig 2.29 AES Key Expansion

Type 6: Key Expansion Algorithm

The AES key expansion algorithm takes as input a four-word (16-byte) key and produces a linear array of 44 words (176 bytes). This is sufficient to provide a four word round key for the initial AddRoundKey stage and each of the 10 rounds of the cipher.

Each added word $w[i]$ depends on the immediately preceding word, $w[i - 1]$, and the word four positions back, $w[i - 4]$. In three out of four cases, a simple XOR is used. For a word whose position in the w array is a multiple of 4, a more complex function is used.

Figure 2.30 illustrates the generation of the expanded key, using the symbol g to represent that complex function. The function g consists of the following sub functions

```

KeyExpansion (byte key[16], word w[44])
{
    word temp
    for (i = 0; i < 4; i++)    w[i] = (key[4*i], key[4*i+1],
                                         key[4*i+2],
                                         key[4*i+3]);
    for (i = 4; i < 44; i++)
    {
        temp = w[i - 1];
        if (i mod 4 = 0)    temp = SubWord (RotWord (temp))
                           + Rcon[i/4];
        w[i] = w[i-4] ⊕ temp
    }
}

```

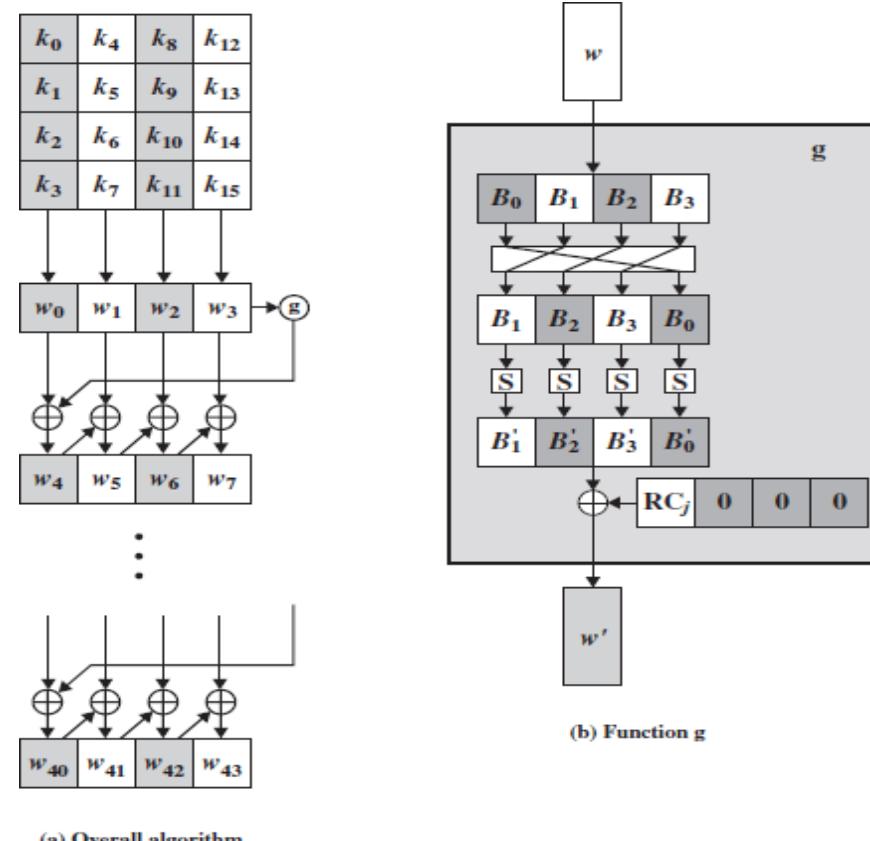


Figure 2.30 Key Expansion Algorithm

1. RotWord performs a one-byte circular left shift on a word. This means that a input word [B0, B1, B2, B3] is transformed into [B1, B2, B3, B0].
2. SubWord performs a byte substitution on each byte of its input word, using the S-box.
3. The result of steps 1 and 2 is XORed with a round constant, Rcon[j].

The round constant is a word in which the three rightmost bytes are always 0. Thus, the effect of an XOR of a word with Rcon is to only perform an XOR on the leftmost byte of the word. The round constant is different for each round and is defined as $Rcon[j] = (RC[j], 0, 0, 0)$, with $RC[1] = 1$, $RC[j] = 2 \# RC[j-1]$ and with multiplication defined over the field GF(28). The values of $RC[j]$ in hexadecimal are

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

For example, suppose that the round key for round 8 is

EA D2 73 21 B5 8D BA D2 31 2B F5 60 7F 8D 29 2F

Then the first 4 bytes (first column) of the round key for round 9 are calculated as follows:

i (decimal)	temp	After RotWord	After SubWord	Rcon (9)	After XOR with Rcon	w[i-4]	w[i] = temp \oplus w[i-4]
36	7F8D292F	8D292F7F	5DA515D2	1B000000	46A515D2	EAD27321	AC7766F3

An AES Example

For this example, the plaintext is a hexadecimal palindrome. The plaintext, key, and resulting ciphertext are

Plaintext:	0123456789abcdeffedcba9876543210
Key:	0f1571c947d9e8590cb7add6af7f6798
Ciphertext:	ff0b844a0853bf7c6934ab4364148fb9

Results

Table 2.4 shows the expansion of the 16-byte key into 10 round keys. The process is formed word by word, with each four-byte word occupying one column of the word round-key matrix.

Key Words	Auxiliary Function
$w_0 = 0f\ 15\ 71\ c9$ $w_1 = 47\ d9\ e8\ 59$ $w_2 = 0c\ b7\ ad\ d6$ $w_3 = af\ 7f\ 67\ 98$	$\text{RotWord}(w_3) = 7f\ 67\ 98\ af = x_1$ $\text{SubWord}(x_1) = d2\ 85\ 46\ 79 = y_1$ $\text{Rcon}(1) = 01\ 00\ 00\ 00$ $y_1 \oplus \text{Rcon}(1) = d3\ 85\ 46\ 79 = z_1$
$w_4 = w_0 \oplus z_1 = dc\ 90\ 37\ b0$ $w_5 = w_4 \oplus w_1 = 9b\ 49\ df\ e9$ $w_6 = w_5 \oplus w_2 = 97\ fe\ 72\ 3f$ $w_7 = w_6 \oplus w_3 = 38\ 81\ 15\ a7$	$\text{RotWord}(w_7) = 81\ 15\ a7\ 38 = x_2$ $\text{SubWord}(x_2) = 0c\ 59\ 5c\ 07 = y_2$ $\text{Rcon}(2) = 02\ 00\ 00\ 00$ $y_2 \oplus \text{Rcon}(2) = 0e\ 59\ 5c\ 07 = z_2$
$w_8 = w_4 \oplus z_2 = d2\ c9\ 6b\ b7$ $w_9 = w_8 \oplus w_5 = 49\ 80\ b4\ 5e$ $w_{10} = w_9 \oplus w_6 = de\ 7e\ c6\ 61$ $w_{11} = w_{10} \oplus w_7 = e6\ ff\ d3\ c6$	$\text{RotWord}(w_{11}) = ff\ d3\ c6\ e6 = x_3$ $\text{SubWord}(x_3) = 16\ 66\ b4\ 83 = y_3$ $\text{Rcon}(3) = 04\ 00\ 00\ 00$ $y_3 \oplus \text{Rcon}(3) = 12\ 66\ b4\ 8e = z_3$
$w_{12} = w_8 \oplus z_3 = c0\ af\ df\ 39$ $w_{13} = w_{12} \oplus w_9 = 89\ 2f\ 6b\ 67$ $w_{14} = w_{13} \oplus w_{10} = 57\ 51\ ad\ 06$ $w_{15} = w_{14} \oplus w_{11} = b1\ ae\ 7e\ c0$	$\text{RotWord}(w_{15}) = ae\ 7e\ c0\ b1 = x_4$ $\text{SubWord}(x_4) = e4\ f3\ ba\ c8 = y_4$ $\text{Rcon}(4) = 08\ 00\ 00\ 00$ $y_4 \oplus \text{Rcon}(4) = ec\ f3\ ba\ c8 = z_4$

Table 2.4 Expansion of the 16-byte key into 10 round keys

The left-hand column shows the four round-key words generated for each round. The right-hand column shows the steps used to generate the auxiliary word used in key expansion. The key itself serving as the round key for round 0.

Next, Table 2.5 shows the progression of **State** through the AES encryption process. The first column shows the value of **State** at the start of a round. For the first row, **State** is just the matrix arrangement of the plaintext. The second, third, and fourth columns show the value of **State** for that round after the SubBytes, ShiftRows, and MixColumns transformations, respectively. The fifth column shows the roundkey.

Key Words	Auxiliary Function
$w_{16} = w_{12} \oplus z_4 = 2c\ 5c\ 65\ f1$ $w_{17} = w_{16} \oplus w_{13} = a5\ 73\ 0e\ 96$ $w_{18} = w_{17} \oplus w_{14} = f2\ 22\ a3\ 90$ $w_{19} = w_{18} \oplus w_{15} = 43\ 8c\ dd\ 50$	$\text{RotWord}(w_{19}) = 8c\ dd\ 50\ 43 = x_5$ $\text{SubWord}(x_5) = 64\ c1\ 53\ 1a = y_5$ $\text{Rcon}(5) = 10\ 00\ 00\ 00$ $y_5 \oplus \text{Rcon}(5) = 74\ c1\ 53\ 1a = z_5$
$w_{20} = w_{16} \oplus z_5 = 58\ 9d\ 36\ eb$ $w_{21} = w_{20} \oplus w_{17} = f1\ ee\ 38\ 7d$ $w_{22} = w_{21} \oplus w_{18} = 0f\ cc\ 9b\ ed$ $w_{23} = w_{22} \oplus w_{19} = 4c\ 40\ 46\ bd$	$\text{RotWord}(w_{23}) = 40\ 46\ bd\ 4c = x_6$ $\text{SubWord}(x_6) = 09\ 5a\ 7a\ 29 = y_6$ $\text{Rcon}(6) = 20\ 00\ 00\ 00$ $y_6 \oplus \text{Rcon}(6) = 29\ 5a\ 7a\ 29 = z_6$
$w_{24} = w_{20} \oplus z_6 = 71\ c7\ 4c\ c2$ $w_{25} = w_{24} \oplus w_{21} = 8c\ 29\ 74\ bf$ $w_{26} = w_{25} \oplus w_{22} = 83\ e5\ ef\ 52$ $w_{27} = w_{26} \oplus w_{23} = cf\ a5\ a9\ ef$	$\text{RotWord}(w_{27}) = a5\ a9\ ef\ cf = x_7$ $\text{SubWord}(x_7) = 06\ d3\ bf\ 8a = y_7$ $\text{Rcon}(7) = 40\ 00\ 00\ 00$ $y_7 \oplus \text{Rcon}(7) = 46\ d3\ df\ 8a = z_7$
$w_{28} = w_{24} \oplus z_7 = 37\ 14\ 93\ 48$ $w_{29} = w_{28} \oplus w_{25} = bb\ 3d\ e7\ f7$ $w_{30} = w_{29} \oplus w_{26} = 38\ d8\ 08\ a5$ $w_{31} = w_{30} \oplus w_{27} = f7\ 7d\ a1\ 4a$	$\text{RotWord}(w_{31}) = 7d\ a1\ 4a\ f7 = x_8$ $\text{SubWord}(x_8) = f1\ 32\ d6\ 68 = y_8$ $\text{Rcon}(8) = 80\ 00\ 00\ 00$ $y_8 \oplus \text{Rcon}(8) = 7f\ 32\ d6\ 68 = z_8$
$w_{32} = w_{28} \oplus z_8 = 48\ 26\ 45\ 20$ $w_{33} = w_{32} \oplus w_{29} = f3\ 1b\ a2\ d7$ $w_{34} = w_{33} \oplus w_{30} = cb\ c3\ aa\ 72$ $w_{35} = w_{34} \oplus w_{32} = 3c\ be\ 0b\ 3$	$\text{RotWord}(w_{35}) = be\ 0b\ 38\ 3c = x_9$ $\text{SubWord}(x_9) = ae\ 2b\ 07\ eb = y_9$ $\text{Rcon}(9) = 1b\ 00\ 00\ 00$ $y_9 \oplus \text{Rcon}(9) = b5\ 2b\ 07\ eb = z_9$
$w_{36} = w_{32} \oplus z_9 = fd\ 0d\ 42\ cb$ $w_{37} = w_{36} \oplus w_{33} = 0e\ 16\ e0\ 1c$ $w_{38} = w_{37} \oplus w_{34} = c5\ d5\ 4a\ 6e$ $w_{39} = w_{38} \oplus w_{35} = f9\ 6b\ 41\ 56$	$\text{RotWord}(w_{39}) = 6b\ 41\ 56\ f9 = x_{10}$ $\text{SubWord}(x_{10}) = 7f\ 83\ b1\ 99 = y_{10}$ $\text{Rcon}(10) = 36\ 00\ 00\ 00$ $y_{10} \oplus \text{Rcon}(10) = 49\ 83\ b1\ 99 = z_{10}$
$w_{40} = w_{36} \oplus z_{10} = b4\ 8e\ f3\ 52$ $w_{41} = w_{40} \oplus w_{37} = ba\ 98\ 13\ 4e$ $w_{42} = w_{41} \oplus w_{38} = 7f\ 4d\ 59\ 20$ $w_{43} = w_{42} \oplus w_{39} = 86\ 26\ 18\ 76$	

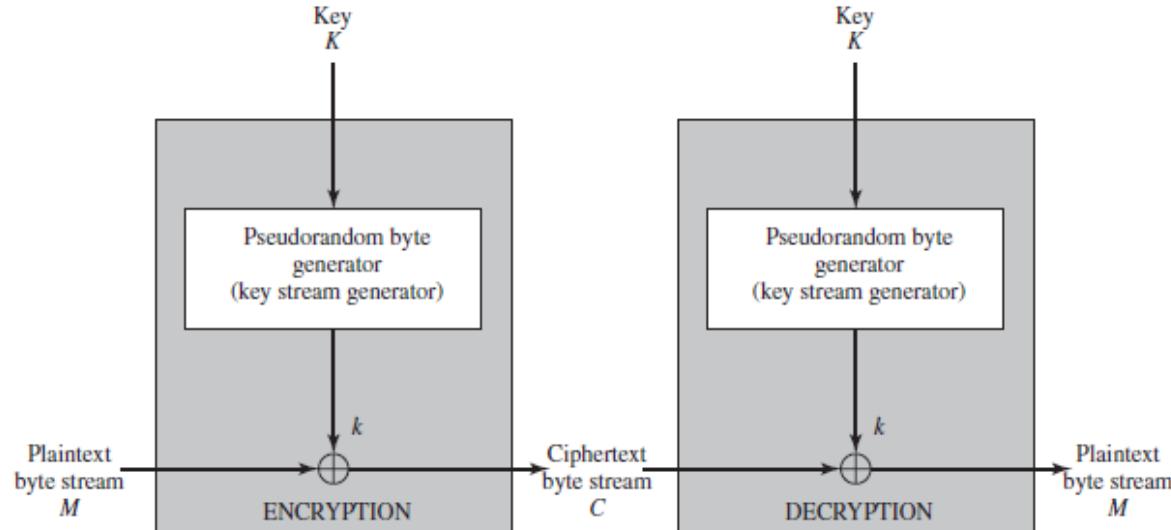
Table 2.5 progression of State through the AES encryption process

2.13 RC4 ALGORITHM

RC4 is an encryption algorithm created in 1987 by Ronald Rivest of RSA Security. It is a stream cipher (figure 2.31), which means that each digit or character is encrypted one at a time. A cipher is a message that has been encoded.

A key input is pseudorandom bit generator that produces a stream 8-bit number that is unpredictable without knowledge of input key.

The output of the generator is called key-stream, is combined one byte at a time with the plaintext stream cipher using X-OR operation.

**Figure 2.31 Stream Cipher Diagram**

Example

RC4 Encryption		RC4 Decryption	
10011000	Plaintext	11001000	Ciphertext
\oplus	01010000 Key Stream	\oplus	01010000 Key Stream
11001000	Ciphertext	10011000	Plaintext

2.13.1 Key Generation Algorithm

A variable-length key from 1 to 256 byte is used to initialize a 256-byte state vector S, with elements S[0] to S[255]. For encryption and decryption, a byte k is generated from S by selecting one of the 255 entries in a systematic fashion, then the entries in S are permuted again (Figure 2.32).

Initialization of S

The entries of S are set equal to the values from 0 to 255 in ascending orders, a temporary vector T, is created. If the length of the key k is 256 bytes, then k is assigned to T. Otherwise, for a key with length(klen) bytes, the first klen elements of T as copied from K and then K is repeated as many times as necessary to fill T.

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

Next, use T to produce the initial permutation of S. Starting with S[0] to S[255], and for each S[i] algorithm swap it with another byte in S according to a scheme dictated by T[i], but S will still contain values from 0 to 255:

```
// Initial Permutation of S
j = 0;
for i = 0 to 255
do
{
j = (j + S[i] + T[i]) mod 256;
Swap(S[i], S[j]);
}
```

Pseudo random generation algorithm (Stream Generation)

Once the vector S is initialized, the input key will not be used. In this step, for each S[i] algorithm swap it with another byte in S according to a scheme dictated by the current configuration of S. After reaching S[255] the process continues, starting from S[0] again

```
//Stream Generation
```

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

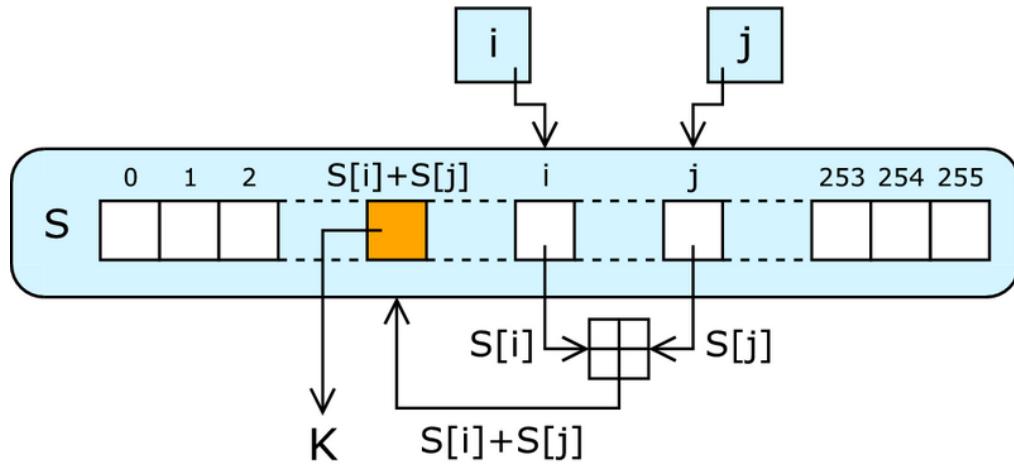


Figure 2.32 PRGA Algorithm

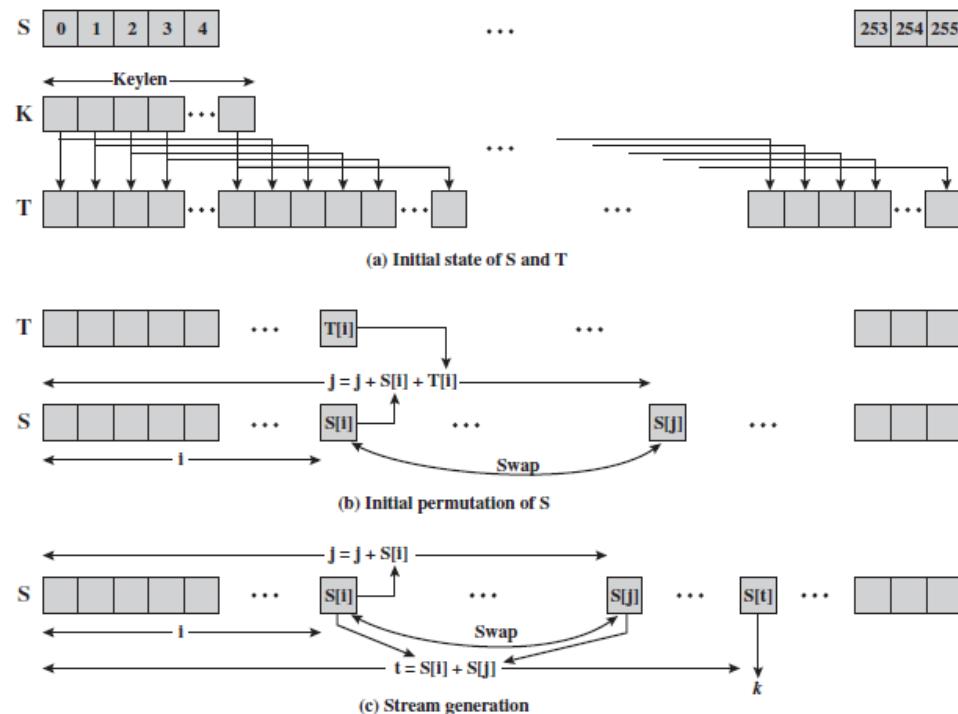
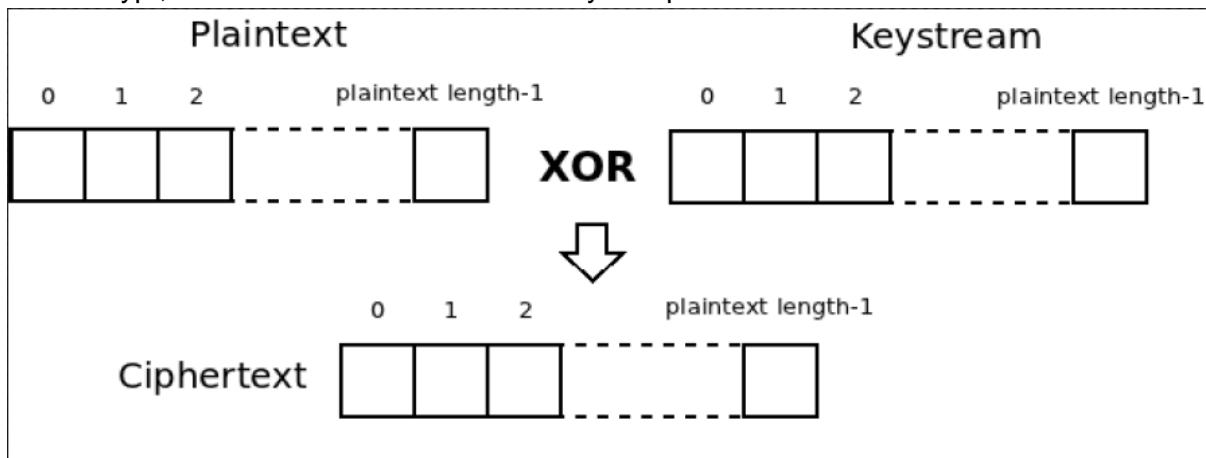


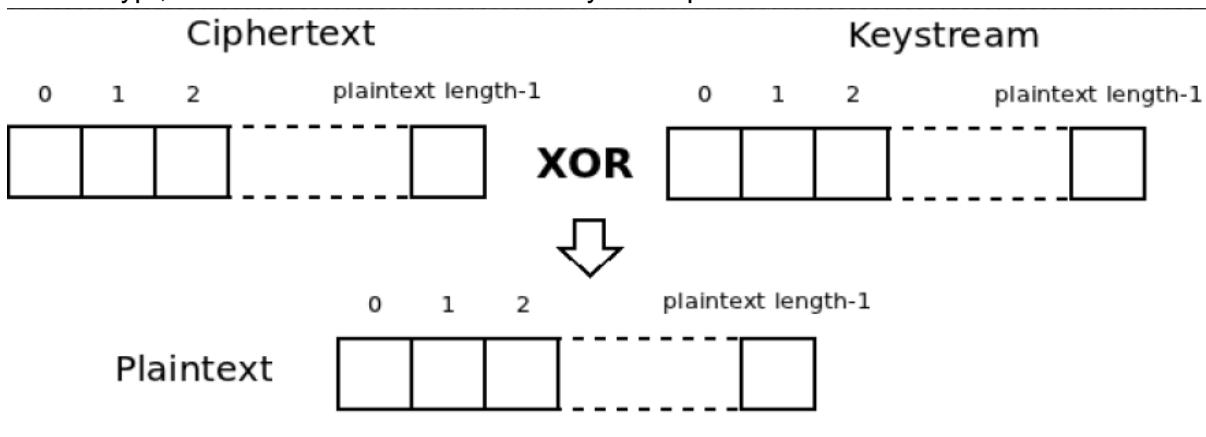
Figure 2.33 RC4 Algorithm

Encrypt using XOR

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

**Figure 2.34 RC4 Encryption****Decrypt using XOR**

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

**Figure 2.35 RC4 Decryption****Advantage**

- It is faster and more suitable for streaming application

2.14 Key Distribution**2.14.1 Symmetric Key Distribution Using Symmetric Encryption**

- In Symmetric key encryption, the two parties to an exchange must share the same key, and that key must be protected from access by others. Therefore, the term that refers to the means of delivering a key to two parties who wish to exchange data, without allowing others to see the key.
- For two parties A and B, key distribution can be achieved in a number of ways, as follows:

1. A can select a key and physically deliver it to B.
2. A third party can select the key and physically deliver it to A and B.
3. If A and B have previously and recently used a key, one party can transmit the new key to the other, encrypted using the old key.
4. If A and B each has an encrypted connection to a third-party C, C can deliver a key on the encrypted links to A and B.

- Physical delivery (1 & 2) is simplest - but only applicable when there is personal contact between recipient and key issuer. This is fine for link encryption where devices & keys occur in pairs, but does not scale as number of parties who wish to communicate grows. 3 are mostly based on 1 or 2 occurring first.
- A third party, whom all parties trust, can be used as a trusted intermediary to mediate the establishment of secure communications between them (4). Must trust intermediary not to abuse the knowledge of all session keys. As numbers of parties grow, some variant of 4 is only practical solution to the huge growth in number of keys potentially needed.

2.14.2 Key Distribution Centre

- The use of a key distribution centre is based on the use of a hierarchy of keys. At a minimum, two levels of keys are used.
- Communication between end systems is encrypted using a temporary key, often referred to as a Session key.
- Typically, the session key is used for the duration of a logical connection and then discarded
- Master key is shared by the key distribution centre and an end system or user and used to encrypt the session key.

2.14.3 Key Distribution Scenario

- Let us assume that user A wishes to establish a logical connection with B and requires a one-time session key to protect the data transmitted over the connection. A has a master key, K_a , known only to itself and the KDC; similarly, B shares the master key K_b with the KDC (Figure 2.36). The following steps occur:

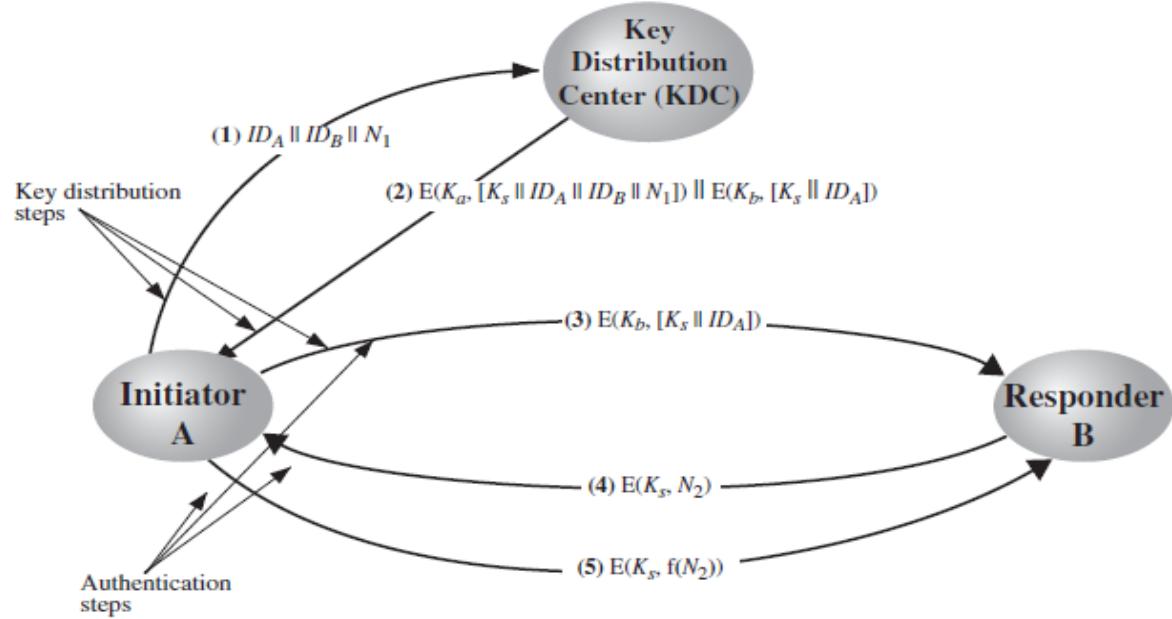


Figure 2.36 Key Distribution Scenarios

1. An issue a request to the KDC for a session key to protect a logical connection to B. The message includes the identity of A and B and a unique identifier, N_1 , for this transaction, which we refer to as a nonce. The nonce may be a timestamp, a counter, or a random number; the minimum requirement is that it differs with each request. Also, to prevent masquerade, it should be difficult for an opponent to guess the nonce. Thus, a random number is a good choice for a nonce.
2. The KDC responds with a message encrypted using K_a . Thus, A is the only one who can successfully read the message, and A knows that it originated at the KDC. The message includes two items intended for A:
 - The one-time session key, K_s , to be used for the session
 - The original request message, including the nonce, to enable A to match this response with the appropriate request

Thus, A can verify that its original request was not altered before reception by the KDC and, because of the nonce, that this is not a replay of some previous request. In addition, the message includes two items intended for B:

- The one-time session key, K_s to be used for the session
- An identifier of A (e.g., its network address), ID_A

These last two items are encrypted with K_b (the master key that the KDC shares with B). They are to be sent to B to establish the connection and prove A's identity.

3. A stores the session key for use in the upcoming session and forwards to B the information that originated at the KDC for B, namely, $E(K_b, [K_s \parallel ID_A])$. Because this information is encrypted with K_b , it is protected from eavesdropping. B now knows the session key (K_s), knows that the other party is A (from ID_A), and knows that the information originated at the KDC (because it is encrypted using K_b). At this point, a session key has been securely delivered to A and B, and they may begin their protected exchange. However, two additional steps are desirable:
 4. Using the newly minted session key for encryption, B sends a nonce, N_2 , to A.
 5. Also using K_s , A responds with $f(N_2)$, where f is a function that performs some transformation on N_2 (e.g., adding one).

2.14.4 Session Key Lifetime

- The distribution of session keys delays the start of any exchange and places a burden on network capacity. A security manager must try to balance these competing considerations in determining the lifetime of a particular session key.
- For connection-oriented protocols, one obvious choice is to use the same session key for the length of time that the connection is open, using a new session key for each new session.
- If a logical connection has a very long lifetime, then it would be prudent to change the session key periodically, perhaps every time the PDU (protocol data unit) sequence number cycles.
- For a connectionless protocol, such as a transaction-oriented protocol, there is no explicit connection initiation or termination.
- Thus, it is not obvious how often one needs to change the session key. The most secure approach is to use a new session key for each exchange.
- A better strategy is to use a given session key for a certain fixed period only or for a certain number of transactions.

UNIT III PUBLIC KEY CRYPTOGRAPHY 9 MATHEMATICS OF ASYMMETRIC KEY CRYPTOGRAPHY

TOPICS: PUBLIC KEY CRYPTOGRAPHY MATHEMATICS OF ASYMMETRIC KEY CRYPTOGRAPHY: Primes – Primality Testing – Factorization – Euler's totient function, Fermat's and Euler's Theorem - Chinese Remainder Theorem – Exponentiation and logarithm
ASYMMETRIC KEY CIPHERS: RSA cryptosystem – Key distribution – Key management – Diffie Hellman key exchange - ElGamal cryptosystem – Elliptic curve arithmetic-Elliptic curve cryptography.

PRIME NUMBERS

Prime numbers

Prime numbers have divisors of 1 and its number itself.

Prime factorisation

To compute **GCD** of any two numbers in prime factorization approach we need to find **prime factors** of the two numbers.

Fermat Theorem or Fermat's little theorem

If a belongs to integer , P is a prime number that does not divide a

then $a^P \equiv a \pmod{P}$

i.e., $a^P \equiv a \pmod{P}$

In special case

$a^{p-1} \equiv 1 \pmod{P}$ if $\text{GCD}(a,P)=1$. where a and p are coprime.

It is mainly used to solve modular exponentiation.

Eg. Compute the value of $2^{10} \pmod{11}$

$$2^{10} \equiv 1 \pmod{11}$$

Eg. Compute the value of $2^{340} \pmod{11}$

$$\begin{aligned} (2^{340}) &= (2^{10})^{34} \pmod{11} \\ &= 1^{34} \pmod{11} \Rightarrow 1 \end{aligned}$$

//Proof.

Take division algorithm

$a = p \cdot q + r$ where can be $0 \leq r \leq p-1$

let g.c.d (a,p) ie coprime

a is not divisible by p hence $1 \leq r \leq p-1$

fact says that if a leaves remainder r where $1 \leq r \leq p-1$ on dividing by p then ka , $1 \leq k \leq p-1$ also leaves remainders from 1 to $p-1$.

Means

If $a, 2a, 3a, \dots, (p-1)a$ surely gives remainders $1, 2, 3, \dots, (p-1)$

So if multiply

$$a^*2a^*3a^*\dots^*(p-1)a \equiv 1.2.3\dots(p-1) \pmod{p}$$

$$\text{hence } a^{p-1} \cdot (p-1)! \equiv (p-1)! \pmod{p}$$

which returns

$$a^{p-1} \equiv 1 \pmod{p} \text{ [as mod } p \text{ cannot divide } (p-1)!!]$$

hence proved. //

Eg. $6^{10} \pmod{11}$

Sol. $6^{11-1} \pmod{11}$

$$= 1 \text{ [as per theorem]}$$

Eg. $5^{15} \pmod{13}$

$$=(5^2 \pmod{13}) * (5^{13} \pmod{13})$$

$$=(25 \pmod{13}) * 5$$

$$=(12 * 5) \pmod{13}$$

$$=60 \pmod{13}$$

$$=8.$$

Euler's theorem

If n and a are coprime positive integers

then $a^{\phi(n)} \equiv 1 \pmod{n}$

In this theorem $\phi(n) = n-1$.

n is prime number and $\phi(n)$ is Euler's phi function.

Euler's phi function is also called Euler's totient function and hence named as Euler's totient theorem or Euler's theorem.

Euler's phi function or Euler's totient function(

Euler's phi function $\phi(n)$ returns the numbers of integers from 1 to n , that are relatively prime to n .

The phi function is computed $\phi(n)$ using various methods. They are

1. If n is a prime number then $\phi(n) = n-1$

2. If n is a composite number then

2.1 Find the prime factors of that number and compute the phi function value as used in step 1. otherwise

2.2. Find prime powers (p^n) of the given number n . For computing the phi value of prime powers we have to use the formula

$$(p^a - p^{a-1}).$$

Eg. Compute Euler's totient function for the values 3, 8

$$1. \phi(3)=3^3-1=2$$

$$2. \phi(8)=2^3$$

$$= 2^3 - 2^{3-1} \quad (\text{since } p^a - p^{a-1}) \\ = 2^3 - 2^2 = 8 - 4 = 4$$

Primality Testing Methods

Primality testing method is a method to find and to prove whether the given number is prime number.

1.Naive Algorithm:

Naïve Algorithm is used to divide the given input number P by all the integers starting from 2 to root of P – 1

If any one of them is a divisor, then the input number P is not a prime. Otherwise, it is considered as a prime number

Algorithm:

1. Pick any integer P that is greater than 2
2. Try to divide P by all integers starting from 2 to the square root of P
3. If P is divisible by any one of these integer, we can conclude that P is a composite
4. Else P is a prime number

Example:

Find the primality test for the number 100 using naïve algorithm.

1. P=100
2. 2,3,4,5,6,7,8,9
3. Case 1: $100/2 = 50$ (composite)

Therefore, 100 is not a prime number.

2.Fermat's Primality Test:

If P is a prime and P does not divide a, which is a natural number then
 $a^{P-1} \equiv 1 \pmod{P}$

Example:

Check whether the given number 12 is prime number or not using Fermat's theorem

Given P = 12

To check whether 12 is prime number or not, we have to check

$$a^{P-1} \equiv 1 \pmod{P}$$

$$a^{12-1} \equiv 1 \pmod{12}$$

$$a^{11} \equiv 1 \pmod{12}$$

Where $1 \leq a < 12$

We have to calculate $a^{11} \pmod{12}$

If it is equal to 1, then it is called prime number. Otherwise, it is called composite number.

Consider, a = 5

$$5^{11} \equiv 1 \pmod{12}$$

$$(i.e) 5^{11} \pmod{12} = 5$$

It is not equal to 1

Therefore it is not a prime number

3.Miller – Rabin Primality Test

Function Miller-Rabin (x)

$$x-1 = (2^w)y \quad // x is the input number for primality test$$

// y is an odd number and 2 is the base

```

select a randomly in the range [2, (x-1)]
Z = ay mod x
if Z congruent 1 (mod x) then return prime
for i = 1 to w - 1
{
  If Z congruent -1 (mod x) then return prime
  Z = Z2 mod x
}
return composite

```

Example:

Find the primality for 7

$x = 7$

As per algorithm, $x - 1 = 7 - 1 = 6 = 2^1 \times 3$

$x = 7$, $w = 1$, $y = 3$

$Z = a^y \text{ mod } 7$

$a = 2$ (randomly), where $[1 \leq a \leq x-1]$

$Z = 2^3 \text{ mod } 7 = 1$

Value of $Z = 1$, 7 is concluded as prime number

Chinese Remainder Theorem

States that when the moduli of a system of linear congruencies are pairwise prime, there is a unique solution of the system modulo, the product of the moduli.

$$x \equiv a \pmod{m}$$

Chinese mathematician Sun Tsu Suan-Ching asking the following problem:

—There are certain things whose number is unknown. When divided by 3, the remainder is 2; when divided by 5, the remainder is 3; and when divided by 7, the remainder is 2. What will be the number of things?

(Otherwise) Mangos are divided into groups consisting of 3 mangos in each group remaining is 2. If the mangos are divided into groups consisting of 5 mangos in each group remaining 3.

If mangos are divided into groups consisting of 7 mangos in each group remaining 2. Totally how many mangos are available?

➤ $x \equiv a_1 \pmod{m_1}$

➤ $x \equiv a_2 \pmod{m_2}$

➤ $x \equiv a_3 \pmod{m_3}$

$$\rightarrow x = \sum (a_i M_i y_i) = (a_1 M_1 y_1 + a_2 M_2 y_2 + a_3 M_3 y_3) \pmod{M}$$

Let M_1, M_2, \dots, M_n be (pairwise) relatively prime numbers. Then the system:

Step1: Calculate M

➤ $M = m_1 * m_2 * m_3 \dots m_n$.

Step 2: Calculate $M_k = M/m_k$

Step 3: Find Inverse of M_k (ie) y_k

Find the X using CRT

➤ $x \equiv 2 \pmod{3}$

➤ $x \equiv 3 \pmod{5}$

➤ $x \equiv 2 \pmod{7}$

$$a_1 = 2, a_2 = 3, a_3 = 2; m_1 = 3, m_2 = 5, m_3 = 7;$$

$$\text{i. } M = m_1 \times m_2 \times m_3 = 105.$$

ii. For each equation calculate

$$M_k = M/m_k \text{ (ie) } M_1 = M / m_1 = 105 / 3 = 35$$

$$M_2 = M / m_2 = 105 / 5 = 21$$

$$M_3 = M / m_3 = 105 / 7 = 15$$

iii. inverse of M_k (ie) y_k

inverse of M_1 (ie) $y_1 = 35^{-1} \pmod{3} = 35^{3-2} \pmod{3} = 2$ [since Fermat's inverse theorem or easy inverse method like $35 \times ? \pmod{3} = 1$ (ie) 2]

$$y_2 = 1; y_3 = 1$$

$$X = \sum(a_i M_i y_i) = (a_1 M_1 y_1 + a_2 M_2 y_2 + a_3 M_3 y_3) \pmod{M}$$

$$X = [(2 \times 35 \times 2) + (21 \times 3 \times 1) + (2 \times 15 \times 1)] \pmod{105}$$

$$= (140 + 63 + 30) \pmod{105}$$

$$= 233 \pmod{105}$$

$$X = 23$$

Exponentiation

- Exponentiation is a type of operation where two elements are used in which one element is considered as a base element and another as an exponential element.
- For example, b is an example of exponential operation where x is a base element and y is an exponential element.
- When y is a positive integer, exponentiation is performed in a similar way to repeated multiplication is performed.
- Modular exponentiation is a type of exponentiation in which a modulo division operation is performed after performing an exponentiation operation.
- For example, $(x^y \pmod{n})$, where n is an integer number.
- The exponentiation is an important concept discussed in many cryptographic algorithms such as RSA, Diffie-Hellman, Elgamal, etc.,

Example:1

$$11^7 \pmod{13}$$

$$11^2 \pmod{13} = 121 \pmod{13} = 4$$

$$11^4 \pmod{13} = (11^2 \pmod{13} \times 11^2 \pmod{13}) \pmod{13}$$

$$= (4 \times 4) \pmod{13}$$

$$= 16 \pmod{13}$$

$$= 3$$

$$11^7 \pmod{13} = (11^4 \pmod{13} \times 11^2 \pmod{13} \times 11^1 \pmod{13}) \pmod{13}$$

$$= (3 \times 4 \times 11) \pmod{13}$$

$$= (132) \pmod{13}$$

$$= 2$$

Find the result of $2^{90} \pmod{13}$.

Solution:

Step 1: Split x and y into smaller parts using exponential rules as shown below:

$$2^{90} \pmod{13} = 2^{50} \times 2^{40}$$

Step 2: Calculate mod n for each part

$$2^{50} \pmod{13} = 1125899906842624 \pmod{13} = 4$$

$$2^{40} \bmod 13 = 1099511627776 \bmod 13 = 3$$

Step 3: Use modular multiplication properties to combine these 2 parts, we have

$$\begin{aligned}2^{90} \bmod 13 &= (2^{50} \times 2^{40}) \bmod 13 \\&= (2^{50} \bmod 13 \times 2^{40} \bmod 13) \bmod 13 \\&= (4 \times 3) \bmod 13 = (12) \bmod 13 = 12\end{aligned}$$

Logarithms or Indices

- Discrete logarithms are logarithms defined with regard to multiplicative cyclic groups. If G is a multiplicative cyclic group and g is a generator of G , then from the definition of cyclic groups, we know every element h in G can be written as g^x for some x . The discrete logarithm to the base g of h in the group G is defined to be x .
- For example, if the group is Z_5^* , and the generator is 2, then the discrete logarithm of 1 is 4 because $2^4 \equiv 1 \pmod{5}$.
- Input: p - prime number, a - primitive root of p , b - a residue mod p .
- Goal: Find k such that $a^k \equiv b \pmod{p}$. (In other words, find the position of y in the large list of $\{a, a^2, \dots, a^{q-1}\}$).
- 14 is a primitive root of 19.
- For example $L_{14}(5) = 10 \bmod 19$, because $14^{10} \equiv 5 \pmod{19}$.
- the inverse problem to exponentiation is to find the **discrete logarithm** of a number modulo p
- that is to find x where $a^x \equiv b \pmod{p}$
- written as $x = \log_a b \bmod p$ or $x = \text{ind}_{a,p}(b)$
- if a is a primitive root then always exists

ASYMMETRIC KEY CIPHERS

PUBLIC KEY CRYPTOGRAPHY:

Principles of public key cryptosystems

The concept of public key cryptography evolved from an attempt to attack two of the most difficult problems associated with symmetric encryption. Key distribution under symmetric key encryption requires either

- (1) Two communicants already share a key, which someone has been distributed to them
- (2) The use of a key distribution center.
 - Digital signatures.

Characteristics of Public key cryptosystems

Public key algorithms rely on one key for encryption and a different but related key for decryption. These algorithms have the following important characteristics:

- It is computationally infeasible to determine the decryption key given only the knowledge of the cryptographic algorithm and the encryption key.

In addition, some algorithms, such as RSA, also exhibit the following characteristic:

- Either of the two related keys can be used for encryption, with the other used for decryption.

INGREDIENTS OF PUBLIC KEY CRYPTOGRAPHY

1. Plaintext: This is the readable message or data that is fed into the algorithm as input.
2. Encryption algorithm: The encryption algorithm performs various transformations on the plaintext.
3. **Public and private keys:** This is a pair of keys that have been selected so that if one is used for encryption, the other is used for decryption. The exact transformations performed by the algorithm depend on the public or private key that is provided as input.
4. Ciphertext: This is the scrambled message produced as output. It depends on the plaintext and the key. For a given message, two different keys will produce two different cipher texts.
5. Decryption algorithm: This algorithm accepts the ciphertext and the matching key and produces the original plaintext.

Encryption:

The essential steps are the following:

1. Each user generates a pair of keys to be used for encryption and decryption of messages.
2. Each user places one of the two keys in a public register or other accessible file. This is the public key. The companion key is kept private.
3. If A wishes to send a confidential message to B, A encrypts the message using B's public key.
4. When B receives the message, it decrypts using its private key.

With this approach(Fig), all participants have access to public keys and private keys are generated locally by each participant and therefore, need not be distributed.

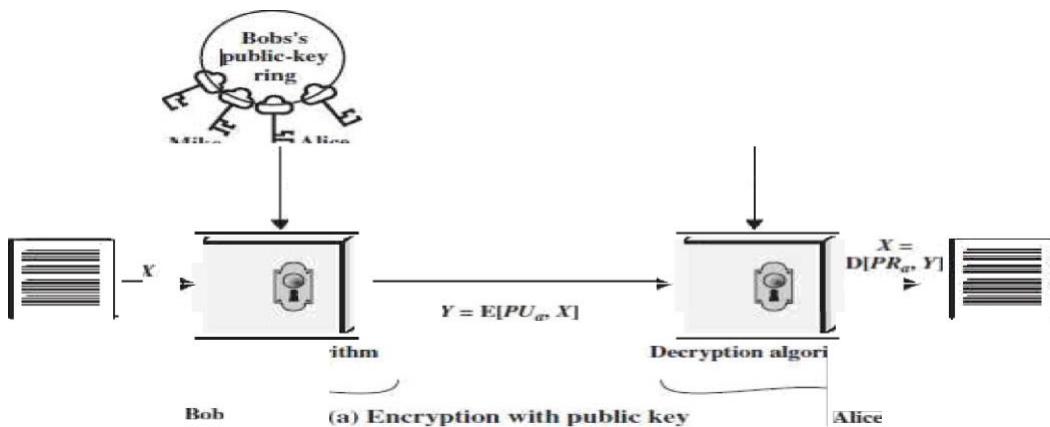


Fig . Public Key Cryptography For Authentication

Let the plaintext be $X=[X_1, X_2, X_3, \dots, X_m]$ where m is the number of letters in some finite alphabets. Suppose A wishes to send a message to B.

B generates a pair of keys: a public key K_{UB} and a private key K_{RB} . K_{RB} is known only to B, whereas K_{UB} is publicly available and therefore accessible by A.

With the message X and encryption key K_{UB} as input, A forms the cipher text $Y=[Y_1, Y_2, Y_3, \dots, Y_n]$.

$$\text{i.e., } Y=EK_{UB}(X)$$

The receiver can decrypt it using the private key KR_b i.e., X=D KR_b(Y)

The other approach (using sender—s private key for encryption and sender—s public key for decryption) will provide authentication which is illustrated in the following diagram(Fig 2.26).

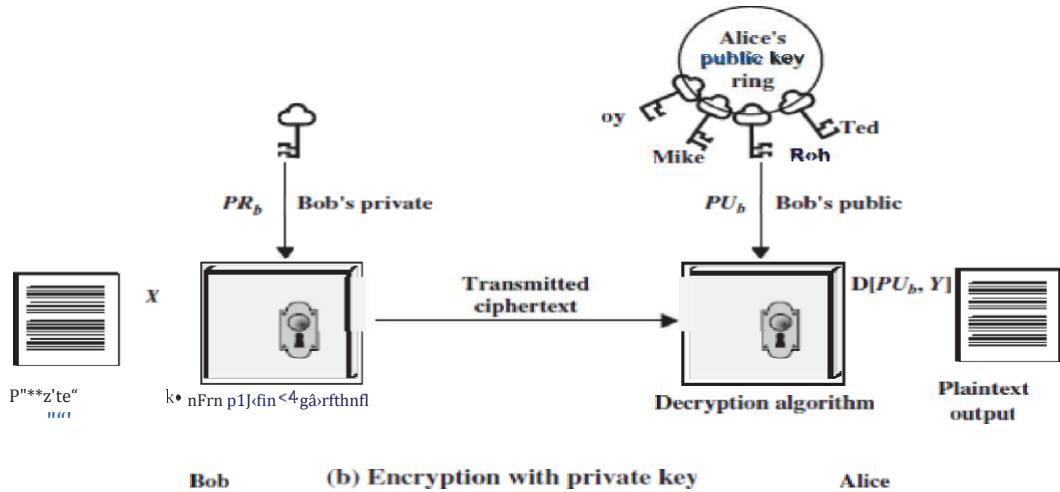


Fig .Private Key Cryptography For Authentication

The encrypted message serves as a digital signature. It is important to emphasize that the encryption process just described does not provide confidentiality.

Differences between public key Encryption and conventional Encryption

Conventional Encryption	Public-Key Encryption
<p><i>Needed to Work:</i></p> <ul style="list-style-type: none"> . The same algorithm uses the same key is used for encryption and decryption. *. The sender and receiver must share the same key. <p><i>/VceJed fnr SeciJriy.-</i></p> <ol style="list-style-type: none"> 1. The key must be kept secret. 2. It must be impractical or at least impractical to decipher a message if the key is kept secret. 3. Knowledge of the algorithm plus one of the keys is sufficient to determine the other key. 	<p><i>Needed to Work:</i></p> <ul style="list-style-type: none"> 1 . One algorithm is used for encryption and a different one for decryption. *. The sender and receiver must each have one of the matched pairs of keys (not the same key). <p>" — "! -'</p> <ol style="list-style-type: none"> 1. One of the keys must be kept secret. *. It must be impossible or at least impractical to decipher a message if one of the keys is known or revealed. 3. Knowledge of the algorithm plus one of the keys plus samples of ciphertext must be insufficient to determine the other key.

Public Key Cryptography for Security

There is some source A that produces a message in plaintext, $X = [X_1, X_2, \dots, X_n]$. The elements of X are letters in some finite alphabet.

The message is intended for destination B. B generates a related pair of keys: a public key, PU_B , and a private key, PR_B . PU_B is known only to B, whereas PR_B is publicly available and therefore accessible by A. With the message and the encryption key PU_B as input, A forms the ciphertext $Y = [Y_1, Y_2, \dots, Y_n]$

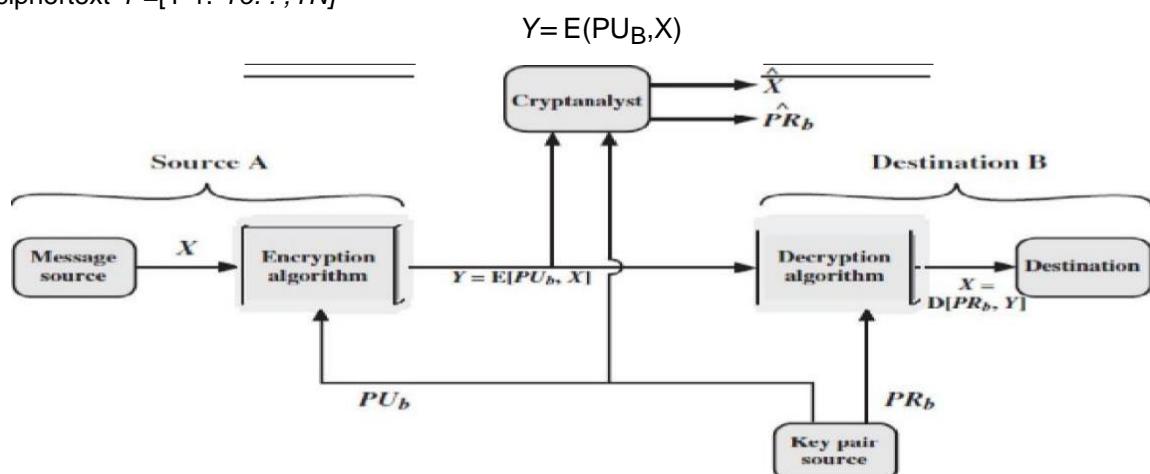


Figure. Public-Key Cryptosystem: Secrecy

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

$$X \xrightarrow{\quad} D(PRd)$$

An adversary, observing Y and having access to PUT , but not having access to PRb or X , must attempt to recover X and/or PRb . It is assumed that the adversary does have knowledge of the encryption (E) and decryption (D) algorithms.

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

$$\begin{aligned} Y &= E(PR_{a1}) \\ X &= D(PU_{a1} \cdot Y) \end{aligned}$$

In this case, A prepares a message to B and encrypts it using A 's private key before transmitting it. B can decrypt the message using A 's public key. Because the message was encrypted using A 's private key, only A could have prepared the message.

Therefore, the entire encrypted message serves as a **digital signature**. In addition, it is impossible to alter the message without access to A 's private key, so the message is authenticated both in terms of source and in terms of data integrity.

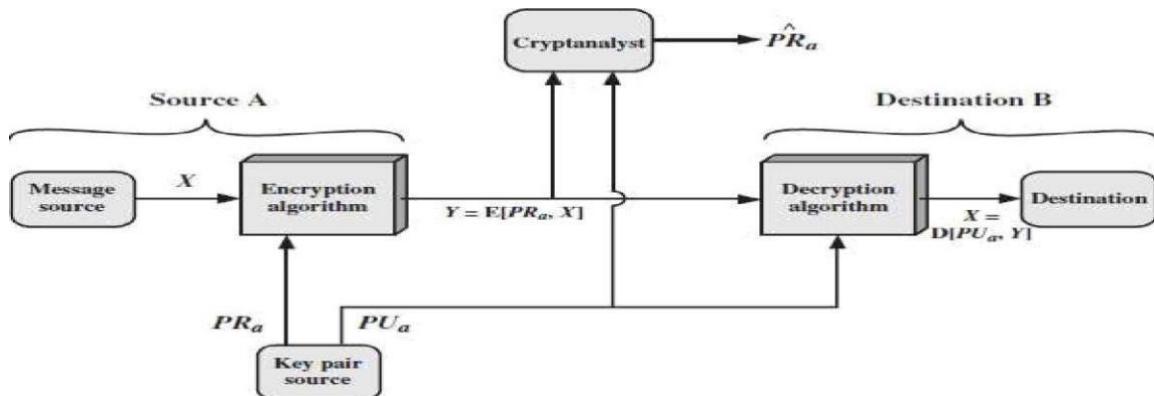


Figure .Public-Key Cryptosystem: Authentication

It is important to emphasize that the encryption process depicted in above Figures does not provide confidentiality. That is, the message being sent is safe from alteration but not from eaves dropping. This is obvious in the case of a signature based on a portion of the message, because the rest of the message is transmitted in the clear. Even in the case of complete encryption, as shown in Figure13, there is no protection of confidentiality because any observer can decrypt the message by using the sender's public key.

Authentication and Secrecy

It is, however, possible to provide both the authentication function and confidentiality by a double use of the public-key scheme (Figure 2.29):

$$\text{Ciphertext } Z = EKUb[EKRa(X)]$$

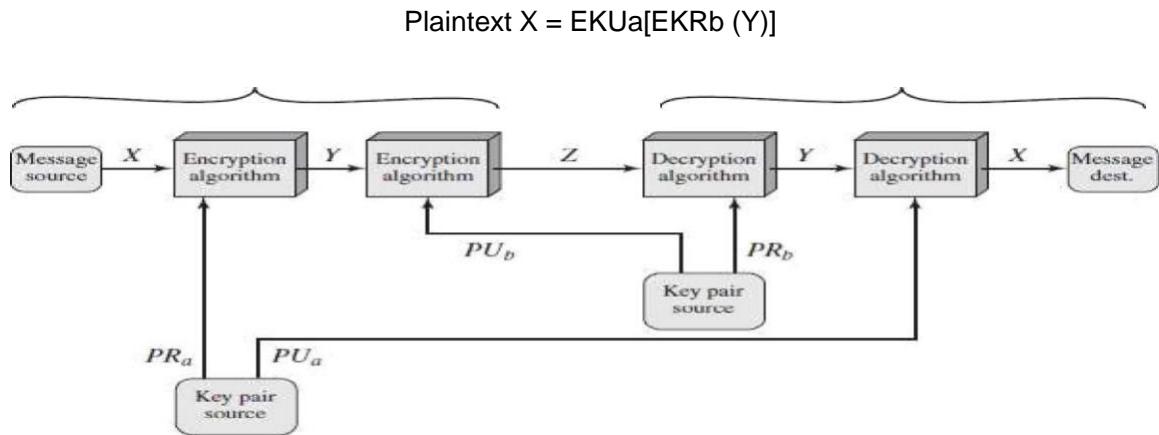


Figure .Public-Key Cryptosystem: Authentication and Secrecy

Initially, the message is encrypted using the sender—s private key. This provides the digital signature. Next, we encrypt again, using the receiver—s public key. The final ciphertext can be decrypted only by the intended receiver, who alone has the matching private key. Thus confidentiality is provided.

Applications for Public-Key Cryptosystems

We can classify the use of public-key cryptosystems into three categories

1. Encryption /decryption: The sender encrypts a message with the recipient's public key.
2. Digital signature: The sender —signs" a message with its private key. Signing is achieved by a cryptographic algorithm applied to the message or to a small block of data that is a function of the message.
3. Key exchange: Two sides cooperate to exchange a session key. Several different approaches are possible, involving the private key(s) of one or both parties.

Requirements for public key cryptography

- It is computationally easy for a party B to generate a pair [KUb ,KRb]
- It is computationally easy for a sender A, knowing the public key and the message to be encrypted M, to generate the corresponding ciphertext: $C=EKUb(M)$.
- It is computationally easy for the receiver B to decrypt the resulting ciphertext using the private key to recover the original message:

$$M = DKRb (C) = DKRb [E KUb (M)]$$

- It is computationally infeasible for an opponent, knowing the public key KUb, to determine the private key KRb
- It is computationally infeasible for an opponent, knowing the public key KUb, and a ciphertext C, to recover the original message M.
- The encryption and decryption functions can be applied in either order:

$$M = EKUb [D KRb (M) = DKUb [E KUb (M)]$$

Public-Key Cryptanalysis

Attack Type 1 :

The public-key encryption scheme is vulnerable to a brute-force attack; therefore use large key. The tradeoff is that makes use of some sort of invertible mathematical function.

Therefore choose key size such that the brute force attack is not possible, at the same time should not be too slow for general use.

Attack type 2:

Attack is of other types (i.e.) given the algorithm and the public key deduce private key. This method has not been successful till date.

Attack Type 3:

A probable-message attack. When a confidential message is to be transmitted using DES, the attacker will find all 2^{56} possible keys using the public key and discover the encrypted key by matching the generated cipher text and the actual cipher. This attack can be avoided by appending some random bits to the message.

RSA ALGORITHM

It was developed by Rivest, Shamir and Adleman. This algorithm makes use of an expression with exponentials. Plaintext is encrypted in blocks, with each block having a binary value less than some number n.

The RSA scheme is a cipher in which the plaintext and cipher text are integers between 0 and $n - 1$ for some n. A typical size for n is 1024 bits, or 309 decimal digits. That is, n is less than 2^{1024}

That is, the block size must be less than or equal to $\log_2(n)$; in practice, the block size is k-bits, where $2^k < n < 2^{k+1}$. Encryption and decryption are of the following form, for some plaintext block M and ciphertext block C:

$$\begin{aligned} c &= M^e \mod n \\ M = C^d \mod n &= (M^e)^d \mod n = M^{ed} \mod n \end{aligned}$$

Both the sender and receiver know the value of n. the sender knows the value of e and only the receiver knows the value of d. thus, this is a public key encryption algorithm with a public key of KU = {e, n} and a private key of KR = {d, n}. For this algorithm to be satisfactory for public key encryption, the following requirements must be met:

1. It is possible to find values of e, d, n such that $M^{ed} \equiv M \pmod{n}$ for all $M < n$.
2. It is relatively easy to calculate M^e and C^d for all values of $M < n$.
3. It is infeasible to determine d given e and n.

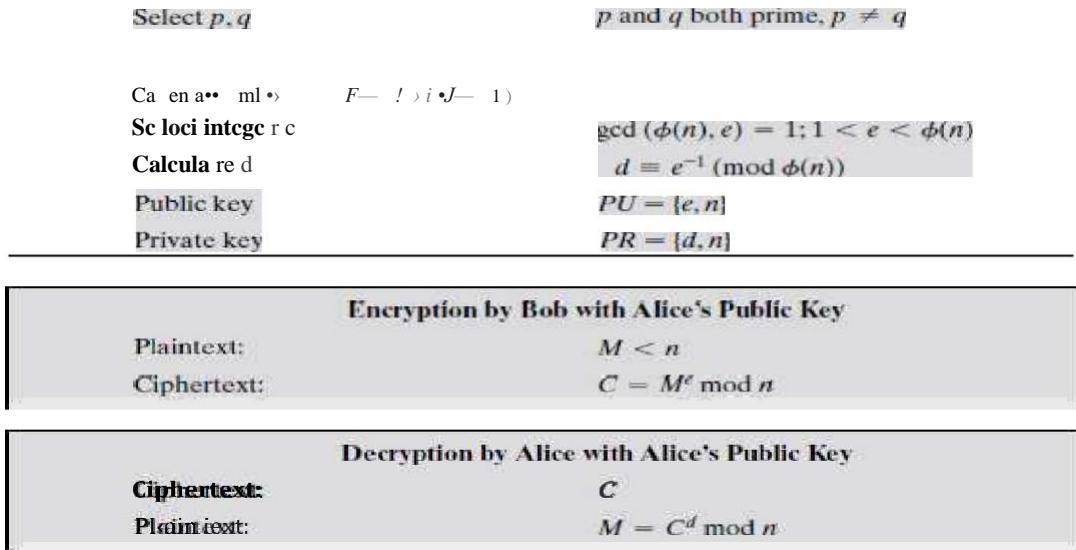


Fig . The RSA Algorithm

Let us focus on the first requirement. We need to find the relationship of the form:

$$M^{ed} = M \pmod{n}$$

Given two prime numbers p and q and two integers, n and m , such that $n=pq$ and $0 < m < n$, and arbitrary integer k , the following relationship holds

$$k6(n)+1 \equiv k(p-1)(q-1)+1 \pmod{n}$$

where $6(n)$ — Euler totient function, which is the number of positive integers less than n and relatively prime to n . we can achieve the desired relationship, if

$$ed = k6(n)+1$$

This is equivalent to saying:

$$\begin{aligned} ed &\equiv 1 \pmod{6(n)} \\ d &\equiv e^{-1} \pmod{6(n)} \end{aligned}$$

That is, e and d are multiplicative inverses mod $6(n)$. According to the rule of modular arithmetic, this is true only if d (and therefore e) is relatively prime to $6(n)$. Equivalently, $\gcd(6(n), d) = 1$.

We are now ready to state the RSA scheme. The ingredients are the following:

p, q , two prime numbers	(private, chosen)
$n = pq$	(public, calculated)
e , with $\gcd(6(n), e) = 1; 1 < e < 6(n)$	(public, chosen)
$d = e^{-1} \pmod{6(n)}$	(private, calculated)

The steps involved in RSA algorithm for generating the key are

- Select two prime numbers, $p = 17$ and $q = 11$.

- Calculate $n = p \cdot q = 17 \cdot 11 = 187$
- Calculate $\phi(n) = (p-1)(q-1) = 16 \cdot 10 = 160$.
- Select e such that e is relatively prime to $\phi(n) = 160$ and less than $\phi(n)$; we choose $e = 7$.
- Determine d such that $de \equiv 1 \pmod{160}$ and $d < 160$. The correct value is $d = 23$, because $23 \cdot 7 = 161 = (1 \cdot 160) + 1$; d can be calculated using the extended Euclid's algorithm

The resulting keys are public key $PU = \{7, 187\}$ and private key $PR = \{23, 187\}$.

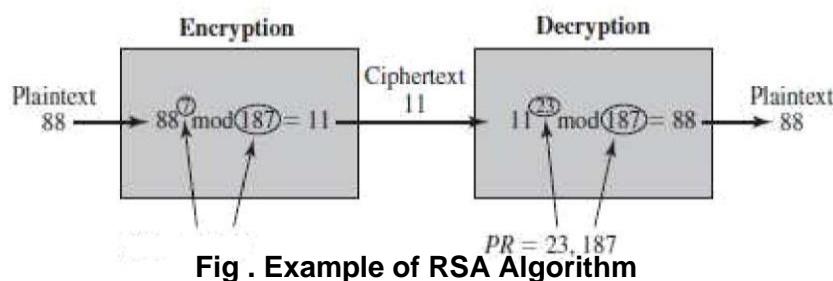
The example shows the use of these keys for a plaintext input of $M = 88$. For encryption, we need to calculate $C = 88^7 \pmod{187}$.

Exploiting the properties of modular arithmetic, we can do this as follows.

$$\begin{aligned} 88^7 \pmod{187} &= [(88^4 \pmod{187}) \cdot (88^2 \pmod{187}) \cdot (88^1 \pmod{187})] \pmod{187} \\ 88^1 \pmod{187} &= 88 \\ 88^2 \pmod{187} &= 7744 \pmod{187} = 77 \\ 88^4 \pmod{187} &= 59,969,536 \pmod{187} = 132 \\ 88^7 \pmod{187} &= (88 \cdot 77 \cdot 132) \pmod{187} = 894,432 \pmod{187} = 11 \end{aligned}$$

For decryption, we calculate $M = 11^{23} \pmod{187}$:

$$\begin{aligned} 11^{23} \pmod{187} &= [(11^1 \pmod{187}) \cdot (11^2 \pmod{187}) \cdot (11^4 \pmod{187}) \cdot (11^1 \pmod{187}) \cdot \\ &(11 \pmod{187})] \pmod{187} \\ 11 \pmod{187} &= 11 \\ 11^1 \pmod{187} &= 121 \\ 11^2 \pmod{187} &= 14,641 \pmod{187} = 55 \\ 11^4 \pmod{187} &= 214,358,881 \pmod{187} = 33 \\ 11^{23} \pmod{187} &= (11 \cdot 121 \cdot 55 \cdot 33 \cdot 33) \pmod{187} = 79,720,245 \pmod{187} = 88 \end{aligned}$$



Security of RSA:

There are three approaches to attack the RSA:

1. Brute force: This involves trying all possible private keys.
2. Mathematical attacks: There are several approaches, all equivalent in effort to factoring the product of two primes.
3. Timing attacks: These depend on the running time of the decryption algorithm.

Type 1 RSA Attack: Defense to Brute Force attack:

Use large key space (i.e) large number of bits in e and d the better secured but problems are,

1. Increases computing power
2. Factoring Problem

Type 2 RSA Attack: Mathematical Attack:

Mathematical approach takes 3 forms:

- Factor $n = p \cdot q$, hence find $\phi(n)$ and then d .
- Determine $\phi(n)$ directly without determining p and q and find
 $d = e^{-1} \pmod{\phi(n)}$
- Find d directly, without first determination $\phi(n)$.

Type 3 RSA Attack: Timing attacks:

This attack is learning for 2 reasons

1. Comes completely from unexpected direction
2. Cipher text only attack

Attack:

If the system does mostly the modular multiplication in majority of cases but takes longer time in few cases. The average is also longer.

The attack is done bit by bit

Start with left most bit b_i ,

Suppose first j bits are known.

For a given cipher text the attacker completes the j iteration.

If the bit is set then $d \leftarrow (d * a) \pmod{n}$.

Methods to overcome Timing attacks:

1. Constant exponentiation time: All exponentiations take the same amount of time before returning a result. This is a simple fix but does degrade performance.
2. Random delay: Better performance could be achieved by adding a random delay to the exponentiation algorithm to confuse the timing attack.
3. Blinding: Multiply the cipher text by a random number before performing exponentiation. This process prevents the attacker from knowing what cipher text bits are being processed inside the computer and therefore prevents the bit-by-bit analysis essential to the timing attack.

KEY MANAGEMENT

There are two uses of public key cryptography regarding the issues of key distribution. They are

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

Distribution of Public Keys

Several techniques have been proposed for the distribution of public keys. Virtually all these proposals can be grouped into the following general schemes:

- a) Public announcement
- b) Publicly available directory
- c) Public-key authority
- d) Public-key certificates

(a) Public Announcement of Public Keys

In public-key encryption the public key is public. Thus, if there is some broadly accepted public-key algorithm, such as RSA, any participant can send his or her public key to any other participant or broadcast the key to the community at large as shown in [Figure 2.32](#).



Figure. Uncontrolled Public-Key Distribution

Disadvantage:

Anyone can forge such a public announcement. That is, some user could pretend to be user A and send a public key to another participant or broadcast such a public key.

Until such time as user A discovers the forgery and alerts other participants, the forger is able to read all encrypted messages intended for A and can use the forged keys for authentication.

(b) Publicly Available Directory

A greater degree of security can be achieved by maintaining a publicly available dynamic directory of public keys. Maintenance and distribution of the public directory would have to be the responsibility of some trusted entity or organization as shown in [Figure 2.33](#). Such a scheme would include the following elements:

1. The authority maintains a directory with a {name, public key} entry for each participant.
2. Each participant registers a public key with the directory authority. Registration would have to be in person or by some form of secure authenticated communication.
3. A participant may replace the existing key with a new one at any time, due to either the key has been used for a large amount of data, or the corresponding private key has been compromised in some way.
4. Participants could also access the directory electronically. For this purpose, secure, authenticated communication from the authority to the participant is mandatory

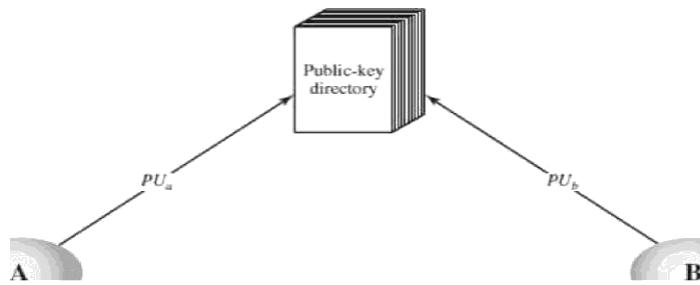


Figure Public-Key Publications

Vulnerabilities:

- 6 Tamper the records of public key directories.
- 6 If an adversary succeeds in obtaining or computing the private key of the directory authority, the adversary could authoritatively pass out counterfeit public keys and impersonate any participant and eavesdrop on messages sent to any participant.

(c) Public-Key Authority

Stronger security for public-key distribution can be achieved by providing tighter control over the distribution of public keys from the directory. A typical scenario is illustrated in Figure 2.34.

As before, the scenario assumes that a central authority maintains a dynamic directory of public keys of all participants. In addition, each participant reliably knows a public key for the authority, with only the authority knowing the corresponding private key. The following steps (matched by number to Figure 2.34) occur:

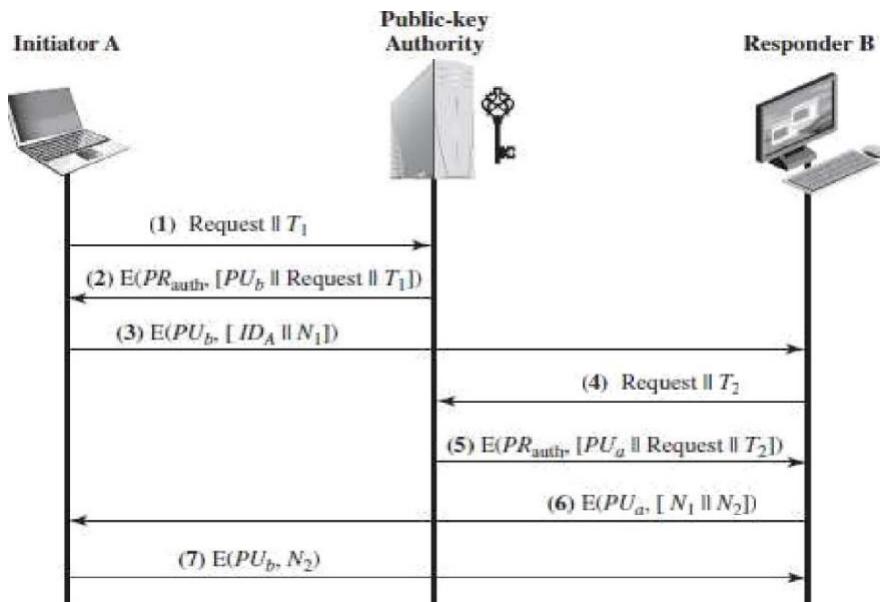


Figure .Public-Key Distribution Scenario

1. A sends a time stamped message to the public-key authority containing a request for the current public key of B.

2. The authority responds with a message that is encrypted using the authority's private key, P_{Rauth} . Thus, A is able to decrypt the message using the authority's public key. Therefore, A is assured that the message originated with the authority. The message includes the following:

- B's public key, PL , which A can use to encrypt messages destined for B
- The original request, to enable A to match this response with the corresponding earlier request and to verify that the original request was not altered before reception by the authority
- The original timestamp, so A can determine that this is not an old message from the authority containing a key other than B's current public key

3. A stores B's public key and also uses it to encrypt a message to B containing an identifier of A (ID_A) and a nonce ($/V_1$), which is used to identify this transaction uniquely.

4. B retrieves A's public key from the authority in the same manner as A retrieved B's public key.

6. B sends a message to A encrypted with K_a and containing A's nonce ($/V_1$) as well as a new nonce generated by B ($/V_2$). Because only B could have decrypted message (3), the presence of $/V_1$ in message (6) assures A that the correspondent is B.

7. A returns 2. encrypted using B's public key, to assure B that its correspondent is A.

Thus, a total of seven messages are required. However, the initial four messages need be used only infrequently because both A and B can save the other's public key for future use, a technique known as caching.

Disadvantages:

- Bottleneck at the authority.

(d) Public-Key Certificates

The scenario of [Figure 2.35](#) is attractive, yet it has some **drawbacks**. The public-key authority could be somewhat of a bottleneck in the system, for a user must appeal to the authority for a public key for every other user that it wishes to contact. As before, the directory of names and public keys maintained by the authority is vulnerable to tampering.

An alternative approach is to use **certificates** that can be used by participants to exchange keys without contacting a public-key authority.

A certificate consists of a public key plus an identifier of the key owner, with the whole block signed by a trusted third party.

A user can present his or her public key to the authority in a secure manner, and obtain a certificate. The user can then publish the certificate. Anyone needing this user's public key can obtain the certificate and verify that it is valid by way of the attached trusted signature.

1. Any participant can read a certificate to determine the name and public key of the certificate's owner.

2. Any participant can verify that the certificate originated from the certificate authority and is not counterfeit.

3. Only the certificate authority can create and update certificates.

These requirements are satisfied by the original proposal in. Denning added the following additional requirement:

4. Any participant can verify the currency of the certificate.

A certificate scheme is illustrated in [Figure](#). Each participant applies to the certificate authority, supplying a public key and requesting a certificate.



Public-Key distribution of Secret Keys using public key cryptography:

- Use previous methods to obtain public-key
- Can use for secrecy or authentication
- Public-key algorithms are slow so usually want to use private-key encryption to protect message contents, Hence need a session key
 - a) Simple
 - b) Secret key distribution with confidentiality and authentication
 - c) Hybrid
 - d) Diffie Hell man key exchange

(a) Simple Secret Key Distribution:

1. A generates a public/private key pair (K_{UA}, K_{RA}) and transmits a message to B consisting of K_{UA} and an identifier of A, ID_A .
2. B generates a secret key K_s , and transmits it to A, encrypted with A's public key.
3. A computes $DK_{RA} [E(K_{UA} [s])]$ to recover the secret key. Because only A can decrypt the message, only A and B will know the identity of K_s -
4. A discards K_{UA} and K_{RA} and B discards K_{UA} .

Advantages:

- No keys exist before the start of the communication no key exist after the completion of communication
- Secure from eaves dropping

Disadvantages:

- Replay attack
- Meet in the middle attack
- A generates a public/private key pair $\{PU_a, PR_a\}$ and transmits a message intended for B consisting of PL/a and an identifier of A, ID_A .
- D intercepts the message, creates its own public/private key pair $\{PU_d, PR_d\}$ and transmits PU_d and ID_A to B.
- B generates a secret key, K_s , and transmits $E(PL/s, K_s)$.
- D intercepts the message and learns K_s by computing $D(PR_d, E(PL/d, K_s))$.
- D transmits $E(PL/d, K_s)$ to A.

(b) Secret Key Distribution with Confidentiality and Authentication:

1. A uses B's public key to encrypt a message to B containing an identifier of A (ID_A) and a nonce (N_1), which is used to identify this transaction uniquely.
2. B sends a message to A encrypted with PU_a and containing A's decrypted message (1), the presence of N_1 in message (2) assures A that correspondent is B.
3. A returns M , encrypted using B's public key, to assure B that its correspondent is A.
4. A selects a secret key s and sends $M = E(K_{UB}[E(K_{RA}[K_s])]$ TO B. Encryption of this message with B's public key ensures that only B can read it; encryption with A's private key ensures that only A could have sent it.
5. Computes $D(PU_b, E(PL/b, K_s))$ to recover the secret key.

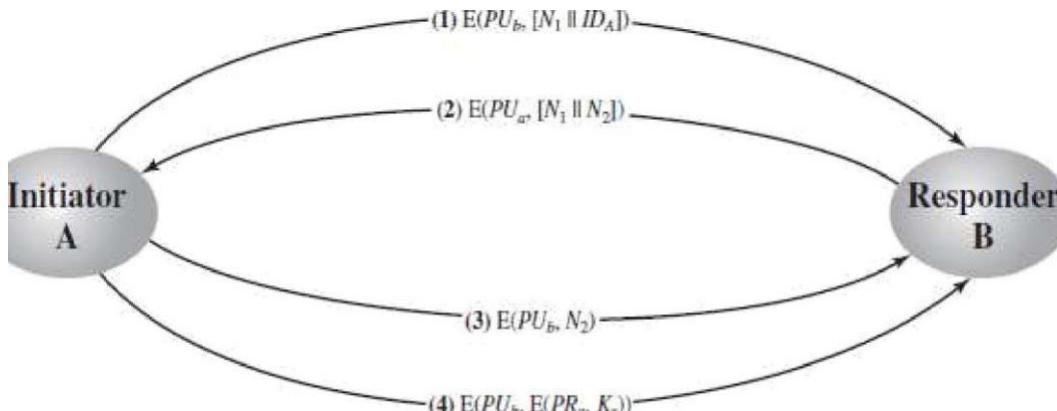


Figure . Public Key Distribution of secret Keys

Advantages:

Scheme ensures both confidentiality and authentication in the exchange of a secret key.

(c) A Hybrid Scheme

Public-key scheme is used to distribute the master keys. The following rationale is provided for using this three-level approach:

1. Performance:

The public key encryption is used occasionally to update the master key between users and KDC.

When the distribution of session keys is done by public key encryption the performance degrades because of high computation needed by P.K.E.

2. **Backward compatibility:** The hybrid scheme is easily overlaid on an existing KDC scheme with minimal disruption or software changes.

The addition of a public-key layer provides a secure, efficient means of distributing master keys.

DIFFIE HELLMAN KEY EXCHANGE

The purpose of the algorithm is to enable two users to exchange a key securely that can then be used for subsequent encryption of messages. The Diffie-Hellman algorithm depends for its effectiveness on the difficulty of computing discrete logarithms.

First, we define a primitive root of a prime number p as one whose power generate all the integers from 1 to $(p-1)$ i.e., if a is a primitive root of a prime number p , then the numbers
 $a \bmod p, a^2 \bmod p, \dots, a^{p-1} \bmod p$

are distinct and consists of integers from 1 to $(p-1)$ in some permutation.

For any integer b and a primitive root a of a prime number p , we can find a unique exponent i such that

$$b = a^i \bmod p \text{ where } 0 \leq i \leq (p-1)$$

The exponent i is referred to as discrete logarithm.

The Algorithm

Figure 2.37 summarizes the Diffie-Hellman key exchange algorithm. There are publicly known numbers: a prime number q and an integer a that is primitive root of q . Suppose users A and B wish to exchange a key. User A selects a random integer $X_A < q$ and computes $Y_A = a^{X_A} \bmod q$.

integer α , such that $\alpha < q$ and
 α is a primitive root of q



Alice and Bob share a prime number q and an integer a such that $a < q$ and a is a primitive root of q

Alice calculates shared secret key $K = (Y_B)^{X_A} \bmod q$

Alice generates a private key X_A such that $X_A < q$

Alice calculates a public key $Y_A = a^{X_A} \bmod q$

Alice receives Bob's public key Y_B in plaintext

Bob calculates shared secret key $K = (Y_A)^{X_B} \bmod q$

Bob generates a private key X_B such that $X_B < q$

Bob calculates a public key $Y_B = a^{X_B} \bmod q$

Bob receives Alice's public key Y_A in plaintext

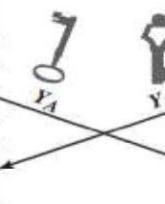


Fig. Diffie Hellman Key Exchange

Similarly, user B independently selects a random integer $X_B < q$ and computes $Y_B = a^{X_B} \bmod q$. Each side keeps the X value private and makes the Y value available publicly to the other side.

User A computes the key as

$$K = (Y_B) \bmod q \text{ and}$$

User B computes the key as

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

These two calculations produce identical results.

$$\begin{aligned} K &= (Y_B) \bmod q \\ &= (a^{X_B} \bmod q) \bmod q \\ &= (a^{X_B}) \bmod q \\ &= (a^X) \bmod q \\ &= (a^X \bmod q)^{X_B} \bmod q \\ &= (Y_A)^{X_B} \bmod q \end{aligned}$$

Global Public Elements	
q	prime number
a	$a < q$ and a is the primitive root of q
User A Key generation	
Select X_A $X_A < q$	
Calculate public Y_A $Y_A = a^{X_A} \bmod q$	
User B Key generation	
Select X_B $X_B < q$	
Calculate public Y_B $Y_B = a^{X_B} \bmod q$	
Generation of secret key by User A	
$K = (Y_B) \bmod q$	
Generation of secret key by User B	
$K = (Y_A)^{X_B} \bmod q$	

The result is that two sides have exchanged a secret key. The security of the algorithm lies in the fact that, while it is relatively easy to calculate exponentials modulo a prime, it is very difficult to calculate discrete logarithms.

Key Exchange Protocols

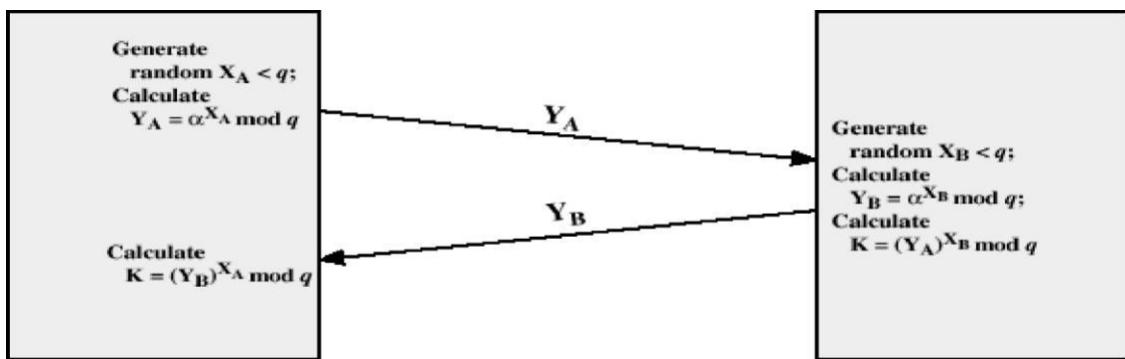


Figure .Diffe-Hellman Key Exchange

The protocol depicted in figure 2.38 is insecure against a man-in-the-middle attack. Suppose Alice and Bob wish to exchange keys, and Darth is the adversary. The attack proceeds as follows:

1. Darth prepares for the attack by generating two random private keys $XD1$ and $XD2$ and then computing the corresponding public keys $YD1$ and $YD2$.
2. Alice transmits YA to Bob.
3. Darth intercepts YA and transmits $YD1$ to Bob. Darth also calculates $= (YA)^{XD} \mod q$.
4. Bob receives $YD1$ and calculates $K1 = (YD1)^{XB} \mod q$.
5. Bob transmits XA to Alice.
6. Darth intercepts XA and transmits $YD2$ to Alice. Darth calculates $1' = (YB)^{XD1} \mod q$.
7. Alice receives $YD2$ and calculates $2' = (YD2)^{XA} \mod q$.

At this point, Bob and Alice think that they share a secret key, but instead Bob and Darth share secret key $K1$ and Alice and Darth share secret key $K2'$. All future communication between Bob and Alice is compromised in the following way:

1. Alice sends an encrypted message $M: E(2, M)$.
2. Darth intercepts the encrypted message and decrypts it, to recover M .
3. Darth sends Bob $E(K1, M)$ or $E(K1, M')$, where M' is any message.

Example:

Key exchange is based on the use of the prime number $q = 353$ and a primitive root of 353, in this case $a = 3$. A and B select secret keys $EA = 97$ and $XB = 233$, respectively.

Each computes its public key:

A computes $YA = 3^{97} \mod 353 = 40$.

B computes $YB = 3^{233} \mod 353 = 248$.

After they exchange public keys, each can compute the common secret key:

A computes $K = (YB)^{XA} \mod 353 = 248^{97} \mod 353 = 160$.

B computes $K = (YA)^{XB} \mod 353 = 40^{233} \mod 353 = 160$.

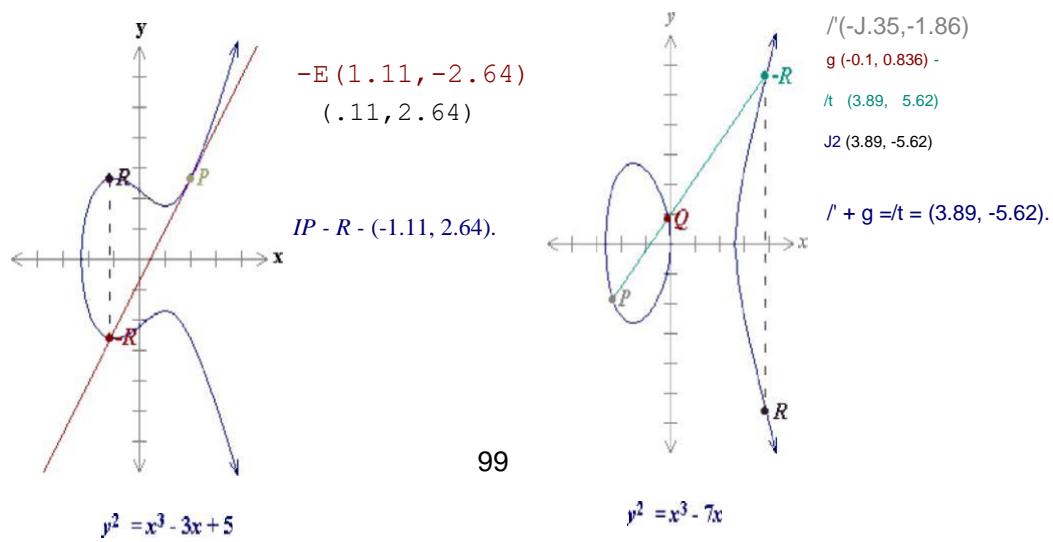
ELLIPTIC CURVE ARITHMETIC

Elliptic Curves:

An elliptic Curve is a Cubic equation of the form

$$y^2 + axy + by = x^3 + cx^2 + dx + e$$

Where a, b, c, d and e are real numbers



A special addition operation is defined over elliptic curves and with the inclusion of a point —0" called point at infinity.

If three points are on a line intersecting an elliptic curve, then their sum is equal to this point at infinity O (which acts as the identity element for this addition operation)

Elliptic Curves over Galois field:

An elliptic group over the Galois Field $Ed(a,b)$ is obtained by computing $x^3+ax+b \bmod p$ for $0 \leq x \leq p$. The constants a&b are non-negative integers smaller than the prime number p must satisfy the condition.

$$4a^3 + 27b^2 \bmod p \neq 0$$

For each value of x, one needs to determine whether or not it is a quadratic residue. If not then the point is not in the elliptic group $Ed(a,b)$

Addition and multiplication operation over elliptic groups:

Let the points $P = (x_1, y_1)$ and $Q = (X_2, Y_2)$ be in the elliptic group $Ed(a,b)$ and O be the point at infinity.

The rules for addition over the elliptic group $Ed(a,b)$ are:

1. $P+O=O+P=P$

2. If $x_2 = x_1$ and $y_2 = -y_1$, that is $P = (x_1, y_1)$ and $Q = (X_2, Y_2) = (x_1, -y_1)$ Then $P+Q = O$

3. If $Q \neq -P$, then their sum $P+Q = (x_3, y_3)$ is given by

$$x_3 = \lambda^2 - x_1 - x_2$$

$$y_3 = \lambda(x_1 - x_3) - y_1$$

and

$$\lambda = \begin{cases} \frac{y_2 - y_1}{x_2 - x_1} & p, \\ \frac{3x + a}{2J} & if P = Q \end{cases}$$

Elliptic Curve Encryption:

Elliptic curve cryptography can be used to encrypt the plain text message M, into ciphertext. The plain text message M is encoded into a point PM from the finite set of points in the elliptic group, $Ed(a,b)$.

The first step consists in choosing a generator point, $G \in Ed(a, b)$, such that the smallest value of n for which $nG = o$ is a very large prime number.

The elliptic group $Ed(a,b)$ and the generator point G are made public.

Each user select a private key, $nA < n$ and compute the public key PA as $PA = nAG$

To encrypt the message point PM for Bob (B),

Alice (A) choses a random integer k and compute the ciphertext pair of points c

Using Bob's public key PB

$$Pc' \quad [(KG), (PM \quad B)]$$

After receiving the ciphertext pair of points, Pc . Bob multiplies the firstpoint,(KG) with his private key nB and then adds the result to the second point in the ciphertext pair of points $(PM + KPB)$

$$(PM + \quad B) - [nB(KG)] = (PM + KnBG) - [nB(KG)] = PM$$

which is the plaintext point, corresponding to the plaintext message M .

Only Bob knowing the private key nB . can remove $nB(KG)$ from the second point of the ciphertext pair of point, i.e $(PM + KGB)$, and hence retrieve the plaintext information PM

Elliptic curve cryptography

Security of ECC:

1. The cryptographic strength of elliptic curve encryption lies in the difficulty for a crypt analyst to determine the secret random number k from KP & P itself.
2. The fastest method to solve this problem (known as elliptic curve logarithm problem is the pollard factorization method).
3. The computational complexity for breaking the elliptic curve cryptosystem, using the pollard method is 3.3×10^{10} MIPS years for an elliptic curve key size of only 150bits.
4. For comparison the fastest method to break RSA, using General Number Field Sieve method to factor the composite integer n in to the two prime p & q requires 2×10^{11} MIPS years for a 768 bit RSA key & 3×10^{11} MIPS years for a RSA key length 1024
5. If the RSA key length is increased to 2048 bits, the GNFS method will need 3×10^{20} MIPS years to factor n whereas increasing the elliptic curve key length to only 24 bits will impose a computational complexity of 1.6×10^2 MIPS years.

Analog of Diffie-Hellman Key Exchange:

Key exchange using elliptic curves can be done in the following manner.

First pick a large integer q , which is either a prime number p or an integer of the form $2^k l$ and elliptic curve parameters a and b . This defines the elliptic group of points $Ed(a, b)$.

Next, pick a base point $G = (x_1, y_1)$ in $Ed(a, b)$ whose order is a very large value n . The order n of a point G on an elliptic curve is the smallest positive integer n such that $nG = O$. $Eq(a, b)$ and G are parameters of the cryptosystem known to all participants.

1. A selects an integer nA less than n . This is A's private key. A then generates a public key PA $nA \times G$; the public key is a point in $Ed(a, b)$.
2. B similarly selects a private key nB and computes a public key PB

3. A generates the secret key $K = nAx$ PB B generates the secret key $K = nB \times EA$.

Ed(a,b) an	Global Public elements Elliptic curve with parameters a,b and q, where q is a prime or integer of the form $2^k + 1$
G	point on elliptic curve whose order is large value n
User A Key Generation	
Select private Calculate public PA	nA $nA < n$ $PA = nA \times G$
User B Key Generation	
Select private Calculate public PA	nB $nB < n$ $PB = nB \times G$
Calculation of secret key by User A	
$K = nAX PB$	
Calculation of secret key by User B	
$K = nBX PA$	

Figure .ECC Diffie-Hellman Key Exchange