UNIT -2

SDES (SIMPLIFIED DES):

- Developed by Professor Edward Schaefer of Santa Clara University
- Has similar properties and structure to DES with much smaller parameters.

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

Five Functions to Encrypt

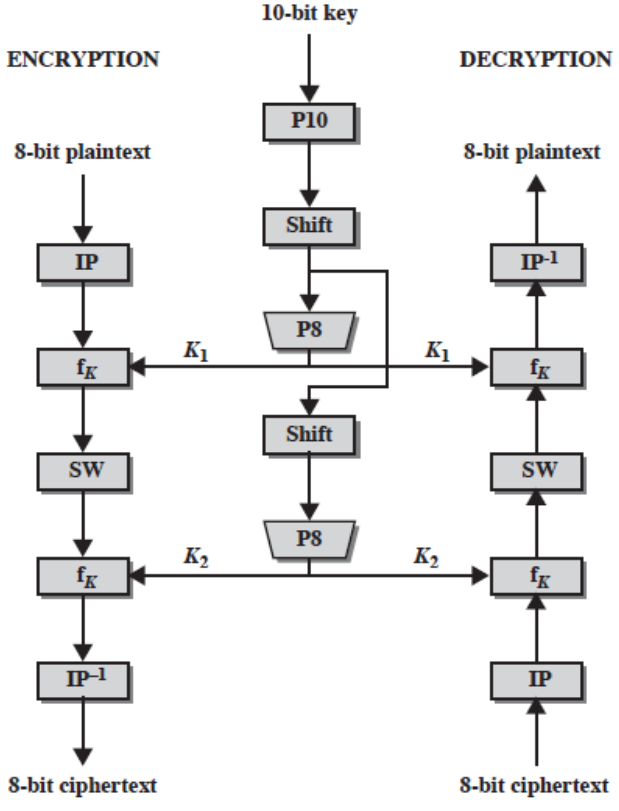
- **IP** – an initial permutation
- $\mathbf{f_k}$ **-** a complex function involves both permutation and substitution.
- **SW** – a simple permutation that swaps the two halves of the data
- $\mathbf{f_k}$ - a complex, 2-input function; again
- $\mathbf{IP^{-1}}$ – inverse permutation of the initial permutation

The **encryption** algorithm can be expressed as

- **Cipher text=** $\mathbf{IP^{-1}.(f_{k2}\,(SW\,(f_{k1}\,(IP\,(plaintext)))))}$
- **k1=p8(shift(p10(key)))**
- **k2=p8(shift(shift(p10(key))))**

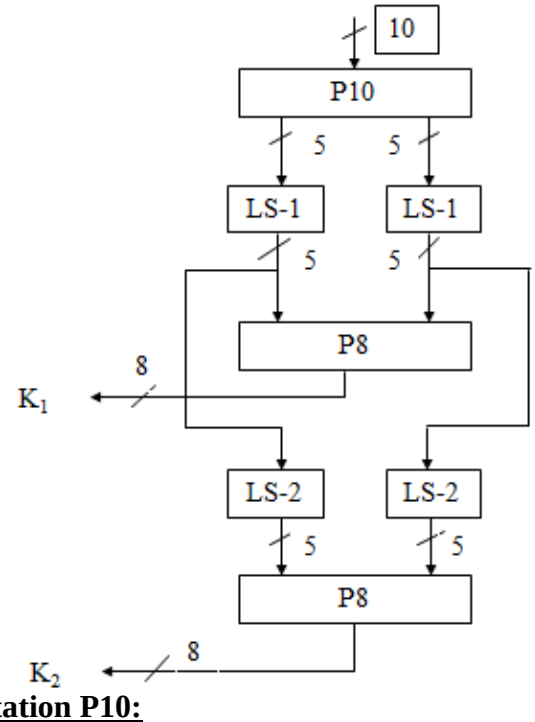
The d**ecryption** algorithm can be expressed as

- **plaintext=** $\mathbf{IP^{-1}.(f_{k1}\,(SW\,(f_{k2}\,(IP\,(cipher\ text)))))}$



SDES KEY GENERATION:

S-DES depends on the use of a 10-bit key shared between sender and receiver. From this key, two 8-bit sub keys are produced for use in particular stages of the encryption and decryption algorithm.



Initial Permutation P10:

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*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.

For example, the key **(1010000010)** is permuted to **(1000001100).**

Circular Left Shift (LS1):

Next, perform a circular left shift (LS-1), or rotation, separately on the first five bits and the second five bits.

In our example, **(10000 01100)** is shifted as **(00001 11000).**

Permutation (P8):

Next we 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

The result is sub key 1 (*K*1).

In our example, **(0000111000)** is permuted as **(10100100).**

Circular Left Shift (LS2):

We 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.

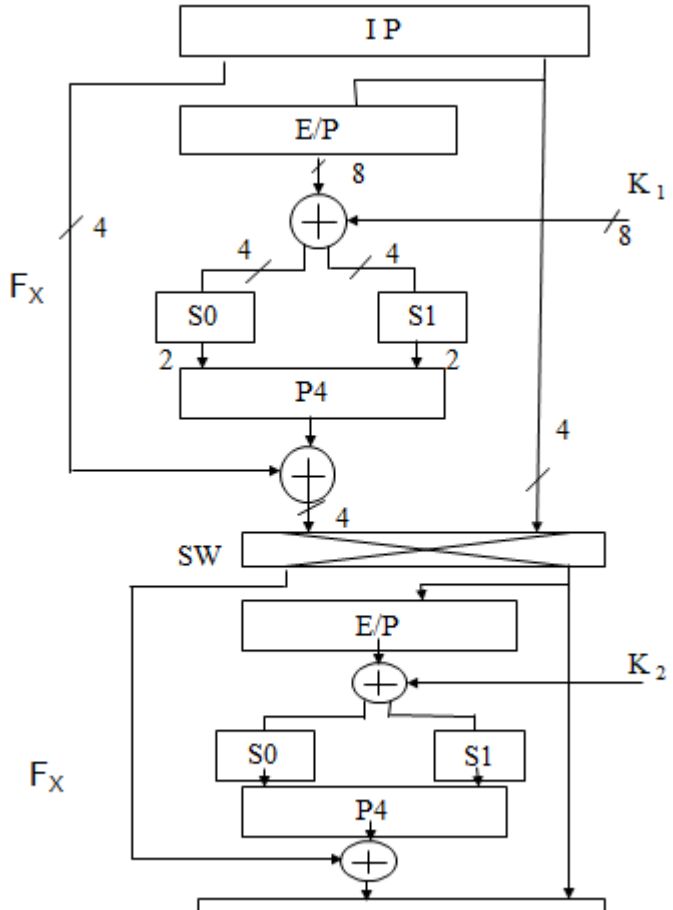
In our example, the value **(00001 11000)** becomes **(00100 00011).**

Permutation (P8):

P8 is applied again to produce *K*2.

In our example, **(00100 00011**) is permuted as **(01000011).**

S-DES ENCRYPTION:



Initial and Final 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

This retains all 8 bits of the plaintext but mixes them up. At the end of the algorithm, the inverse permutation is used:

IP^{-1}							
4	1	3	5	7	2	8	6

The second permutation is indeed the reverse of the first; that is,
IP–1(IP(*X*)) = *X*.

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* -> subkey

$\oplus$ -> bit-by-bit exclusive-OR function.

For example, suppose the output of the is **(10111101)** and **F(1101, *SK*) = (1110)** for some key *SK*. Then **f*K*(10111101) = (01011101)** because **(1011) (1110) = (0101).**

We now describe the mapping F.

Expansion/Permutation:

The input is a 4-bit number ($n_1n_2n_3n_4$).

E/P							
4	1	2	3	2	3	4	1

For what follows, it is clearer to depict the result in this fashion:

n_4	n_1	n_2	n_3
n_2	n_3	n_4	n_1

Exclusive OR:

8-bit sub key *K*1 = (*k*11, *k*12, *k*13, *k*14, *k*15, *k*16, *k*17, *k*18) is added to this value using exclusive-OR:

$n_4 \oplus k_{11}$	$n_1 \oplus k_{12}$	$n_2 \oplus k_{13}$	$n_3 \oplus k_{14}$
$n_2 \oplus k_{15}$	$n_3 \oplus k_{16}$	$n_4 \oplus k_{17}$	$n_1 \oplus k_{18}$

Let us rename these 8 bits:

$p_{0,0}$	$p_{0,1}$	$p_{0,2}$	$p_{0,3}$
$p_{1,0}$	$p_{1,1}$	$p_{1,2}$	$p_{1,3}$

S-BOX:

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:

$$S0 = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 1 & 0 & 3 & 2 \\ 3 & 2 & 1 & 0 \\ 0 & 2 & 1 & 3 \\ 3 & 1 & 3 & 2 \end{bmatrix} \end{array} \qquad S1 = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0 & 1 & 2 & 3 \\ 2 & 0 & 1 & 3 \\ 3 & 0 & 1 & 0 \\ 2 & 1 & 0 & 3 \end{bmatrix} \end{array}$$

The S-boxes operate as follows.

1,4 bits -> a row of the S box

2,3 bits -> a column of the S box.

The entry in that row and column, in base 2, is the 2-bit output.

For example **(0100), (*p*0,0*p*0,3) = (00)** and **(*p*0,1*p*0,2) = (10),** then the output is from row 0, column 2 of S0, which is **3, or (11)** in binary.

Similarly, (*p*10, *p13*) and (*p*11, *p12*) are used to index into a row and column of S1 to produce an additional 2 bits.

Permutation (P4):

The 4 bits produced by S0 and S1 undergo a further permutation as follows:

P4			
2	4	3	1

The output of P4 is the output of the function F.

The Switch Function

The function f_K only alters the leftmost 4 bits of the input. The switch function (SW) Interchanges the left and right 4 bits so that the second instance of f_K operates on a different 4 bits. In this second instance, the E/P, S0, S1, and P4 functions are the same. The key input is K2.

Data Encryption Standard (DES)

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

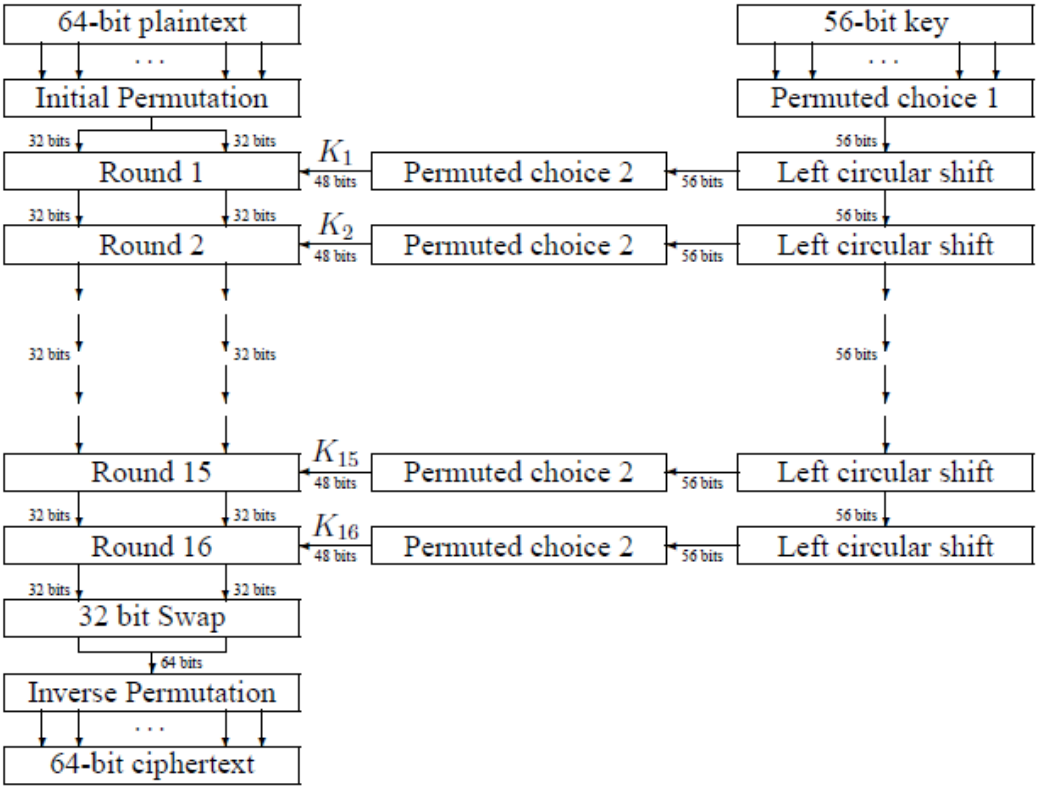
DES History

- IBM developed Lucifer cipher
- by team led by Feistel
- 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 – Basics

- DES uses the two basic techniques of cryptography - confusion and diffusion.
- At the simplest level, diffusion is achieved through numerous permutations and confusions is achieved through the XOR operation.

DES Encryption :



DES - The 16 Rounds:

The basic process in enciphering a 64-bit data block and a 56-bit key using the DES consists of:

- An initial permutation (IP)
- 16 rounds of a complex key dependent calculation f
- A final permutation, being the inverse of IP

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)

(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

(b) 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

(c) 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

(d) 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

DES Round Structure

- uses two 32-bit L & R halves
- as for any Feistel cipher can describe as:
- $Li = Ri–1$
- $Ri = Li–1$ xor F($Ri–1$, Ki)
- takes 32-bit R half and 48-bit subkey and:
    - expands R to 48-bits using perm E
    - adds to subkey
    - passes through 8 S-boxes to get 32-bit result
    - finally permutes this using 32-bit perm P

The use of the key can be seen in the right hand portion of figure.

- • Initially the key is passed through a permutation function (**PC**1)
- • For each of the 16 iterations, a subkey (**K**i) is produced by a combination of a left circular shift and a permutation (**PC**2) which is the same for each iteration. However, the resulting subkey is different for each iteration because of repeated shifts.

(a) Input Key

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

(b) Permuted Choice One (PC-1)

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

(c) Permuted Choice Two (PC-2)

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

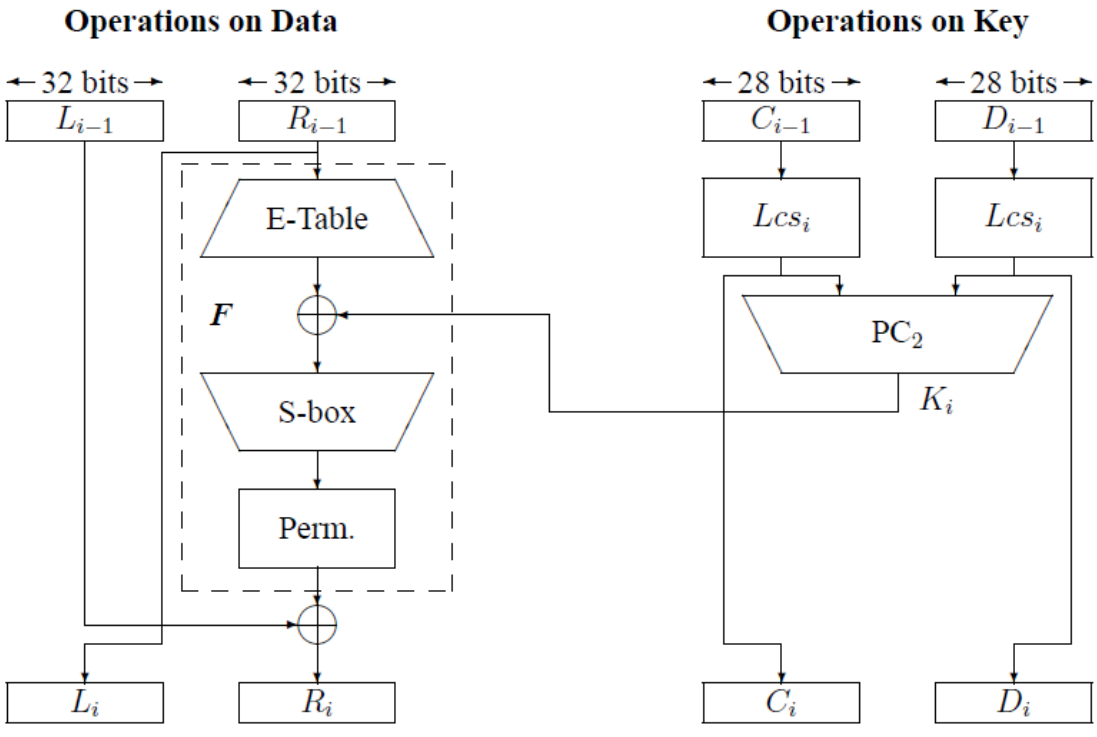
(d) Schedule of Left Shifts

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

The common formulas used to describe the relationships between the input to one round and its output (or the input to the next round) are:

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

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



E – Box Expansion:

The E-box expansion permutation - here the 32-bit input data from Ri−1 is expanded and permuted to give the 48 bits necessary for combination with the 48 bit key. The E-box expansion permutation delivers a larger output by splitting its input into 8, 4-bit blocks and copying every first and fourth bit in each block into the output in a defined manner. The security offered by this operation comes from one bit affecting two substitutions in the S-boxes. This causes the dependency of the output bits on the input bits to spread faster, and is known as the avalanche affect.

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

The bit by bit addition modulo 2 (or exclusive OR) of the E-box output and 48 bit subkey Ki.

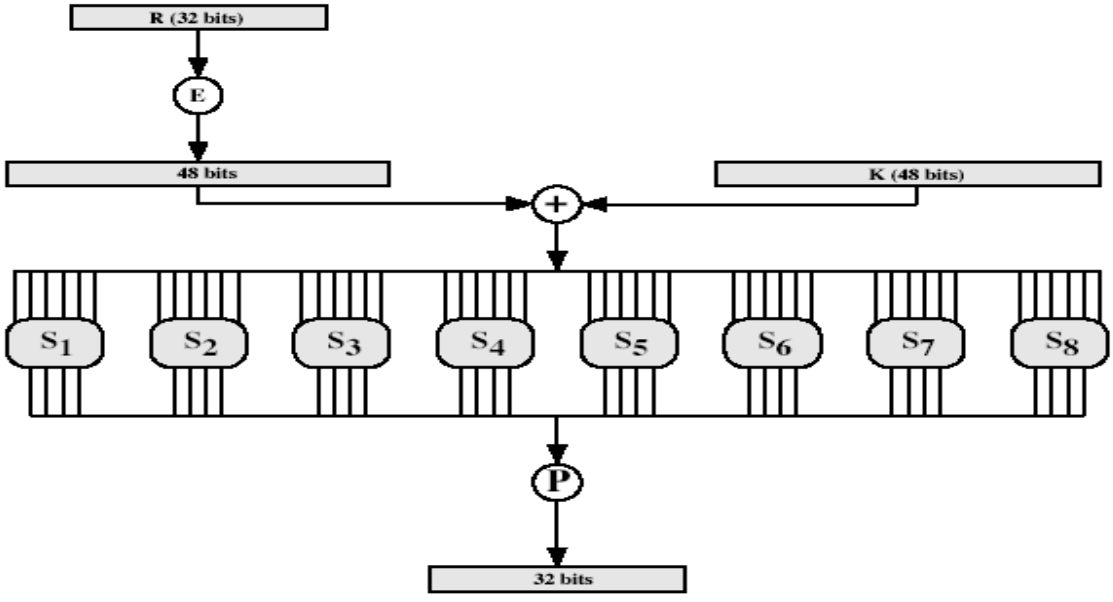
Substitution Boxes S

- 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 rows
- 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: For input 011001

The row is 01(row 1) & column is 1100(col 12),The value in row 1, col 12 is 9

So the output is 1001.



S_1	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13
S_2	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9
S_3	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12
S_4	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14
S_5	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3
S_6	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13
S_7	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12
S_8	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

DES - Swapping of Left and Right Halves

- The 64-bit block being enciphered is broken into two halves.
- The right half goes through one DES round, and the result becomes the new left half.
- The old left half becomes the new right half, and will go through one round in the next round.
- This goes on for 16 rounds, but after the last round the left and right halves are not swapped.

DES Key Schedule

- forms subkeys used in each round
- consists of:
    - initial permutation of the key (PC1) which selects 56-bits in two 28-bit halves
    - 16 stages consisting of:
        - selecting 24-bits from each half
        - permuting them by PC2 for use in function f,
        - rotating each half separately either 1 or 2 places depending on the key rotation schedule K

DES decryption

Decryption uses the same algorithm as encryption except the application of subkeys is reversed (SK16 … SK1).

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.

DES - 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 1990
- 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 Compares Pairs of Encryptions

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

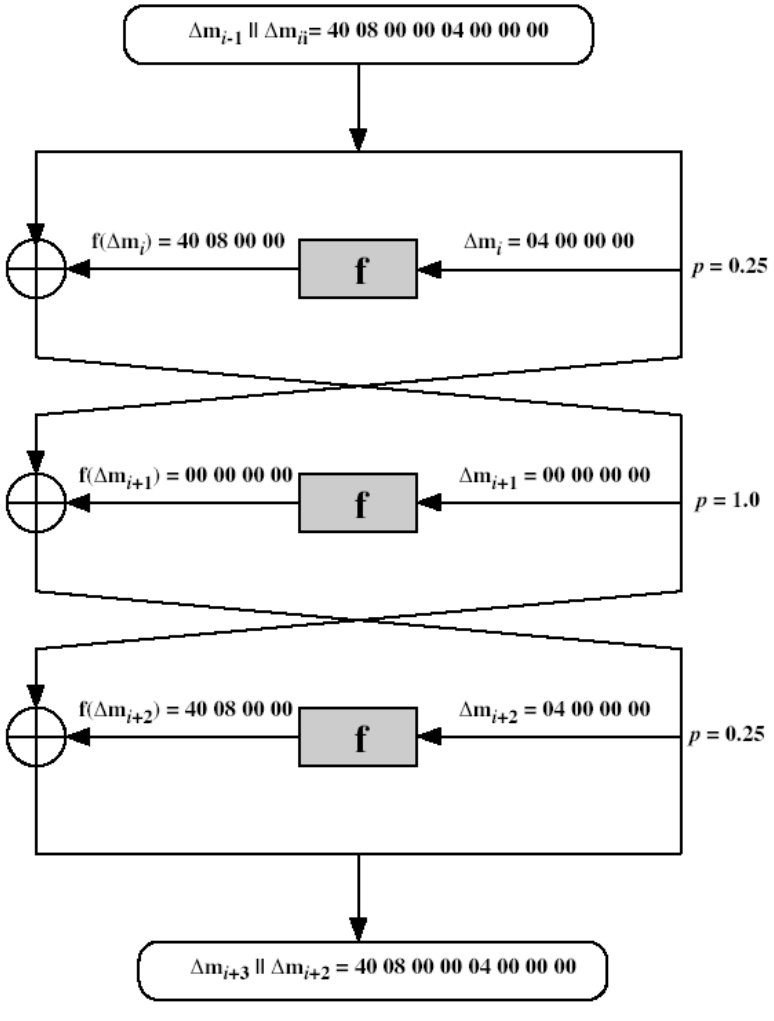
$$\Delta m_{i+1} = m_{i+1} \oplus m'_{i+1}$$

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

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

- 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)
- 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^{47} known plaintexts, still in practise infeasible
- find linear approximations with prob p != ½
- P[i1,i2,...,ia](+)C[j1,j2,...,jb] = K[k1,k2,...,kc]
- where ia,jb,kc 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–½|

BLOCK CIPHER DESIGN PRINCIPLES AND MODES OF OPERATION

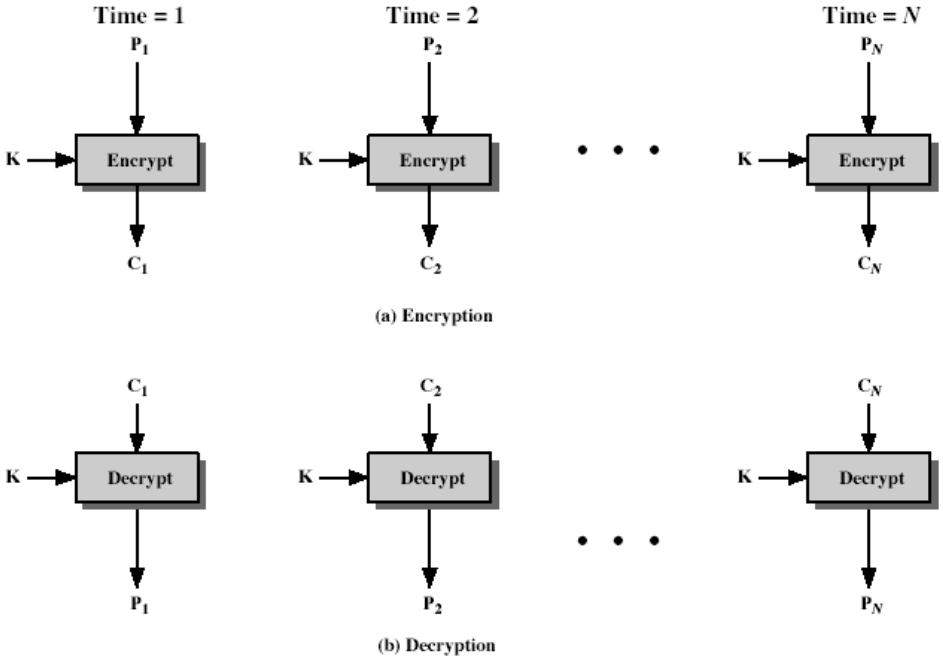
- basic principles still like Feistel in 1970‟s
- number of rounds
- more is better, exhaustive search best attack
- function f:
- provides “confusion”, is nonlinear, avalanche
- key schedule
- complex subkey creation, key avalanche

MODES OF OPERATION

- Mode of operation is a technique for enhancing the effect of a cryptographic algorithm.
- Five Mode of operations are there
  - Electronic code book
  - Cipher block chaining
  - Cipher feedback mode
  - Output feedback mode
  - Counter mode

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
- Ci = DESK1 (Pi)
- uses: secure transmission of single values

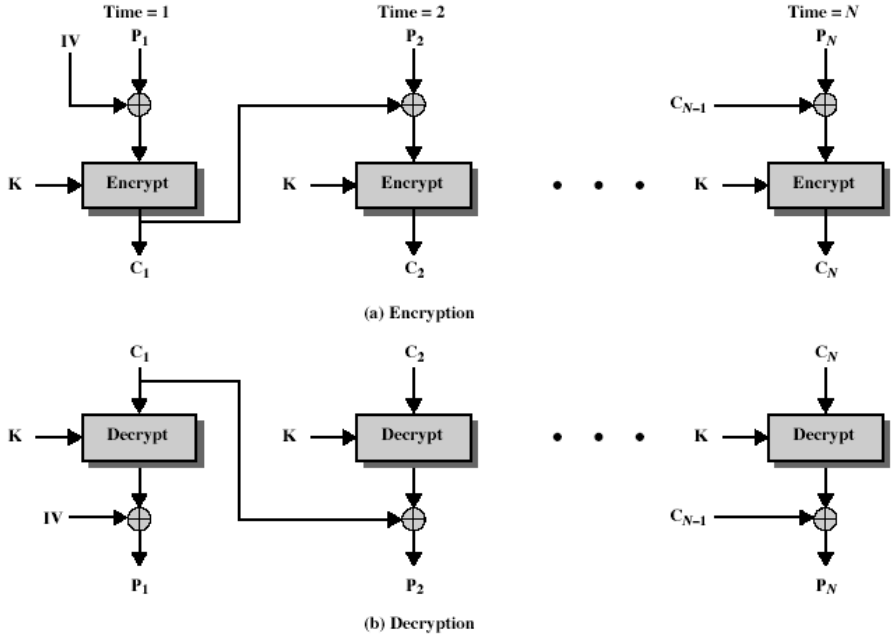


Advantages and Limitations of ECB

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

CIPHER BLOCK CHAINING (CBC)

- message is broken into blocks
- but these are linked together in the encryption operation
- each previous cipher blocks is chained with current plaintext block, hence name
- use Initial Vector (IV) to start process
- Ci = DESK1(Pi XOR Ci-1)
- C-1 = IV
- uses: bulk data encryption, authentication .

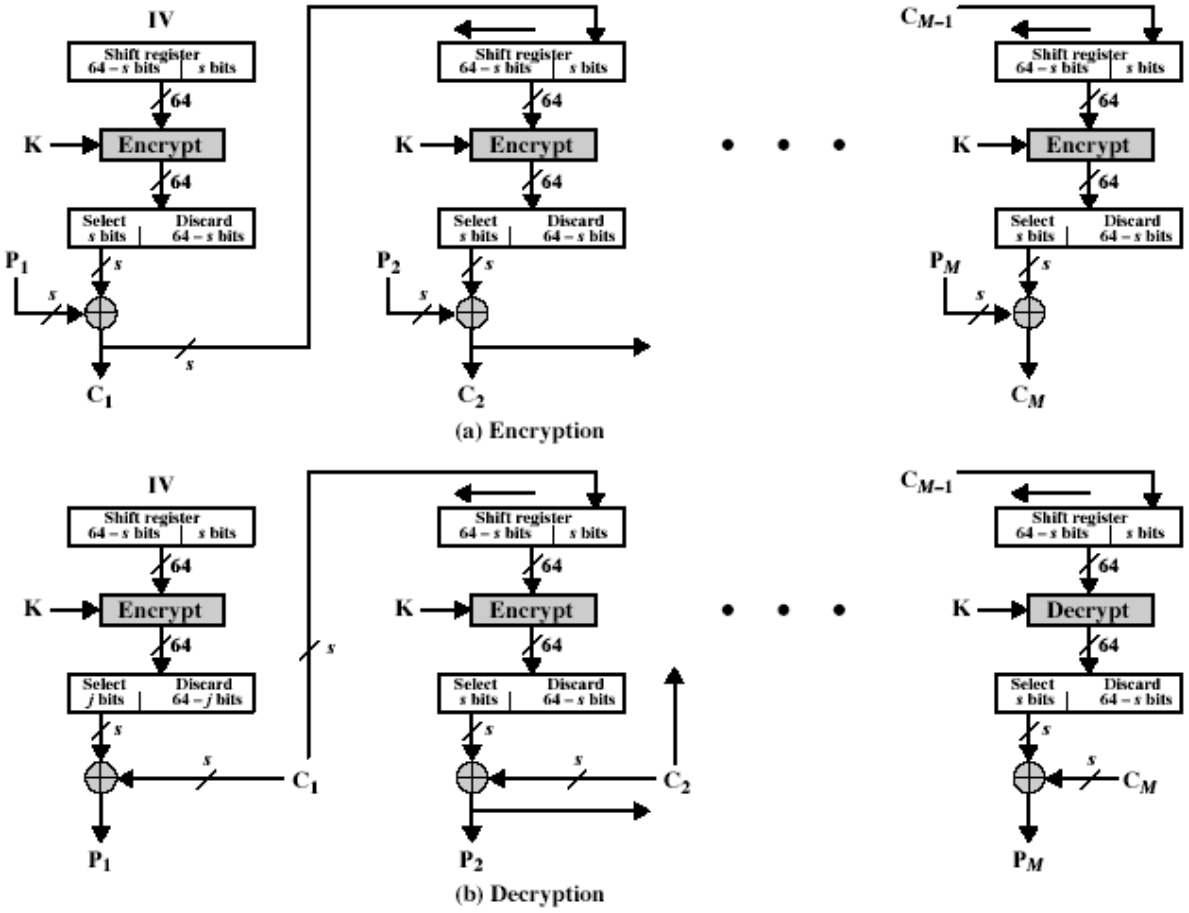


Advantages and Limitations of CBC

- each ciphertext block depends on all message blocks
- thus a change in the message affects all ciphertext blocks after the change as well as the original block
- need Initial Value (IV) known to sender & receiver
    - however if IV is sent in the clear, an attacker can change bits of the first block, and change IV to compensate
    - hence either IV must be a fixed value (as in EFTPOS) or it must be sent encrypted in ECB mode before rest of message
- at end of message, handle possible last short block
    - by padding either with known non-data value (eg nulls)
    - or pad last block with count of pad size
        - eg. [b1 b2 b3 0 0 0 0 5] <- 3 data bytes, then 5 bytes pad+count

CIPHER FEEDBACK (CFB)

- message is treated as a stream of bits
- added to the output of the block cipher
- result is feed back for next stage (hence name)
- standard allows any number of bit (1,8 or 64 or whatever) to be feed back
- denoted CFB-1, CFB-8, CFB-64 etc
- is most efficient to use all 64 bits (CFB-64)
  $Ci = Pi \text{ XOR } DES_{K1}(C_{i-1})$
  C-1 = IV
- uses: stream data encryption, authentication

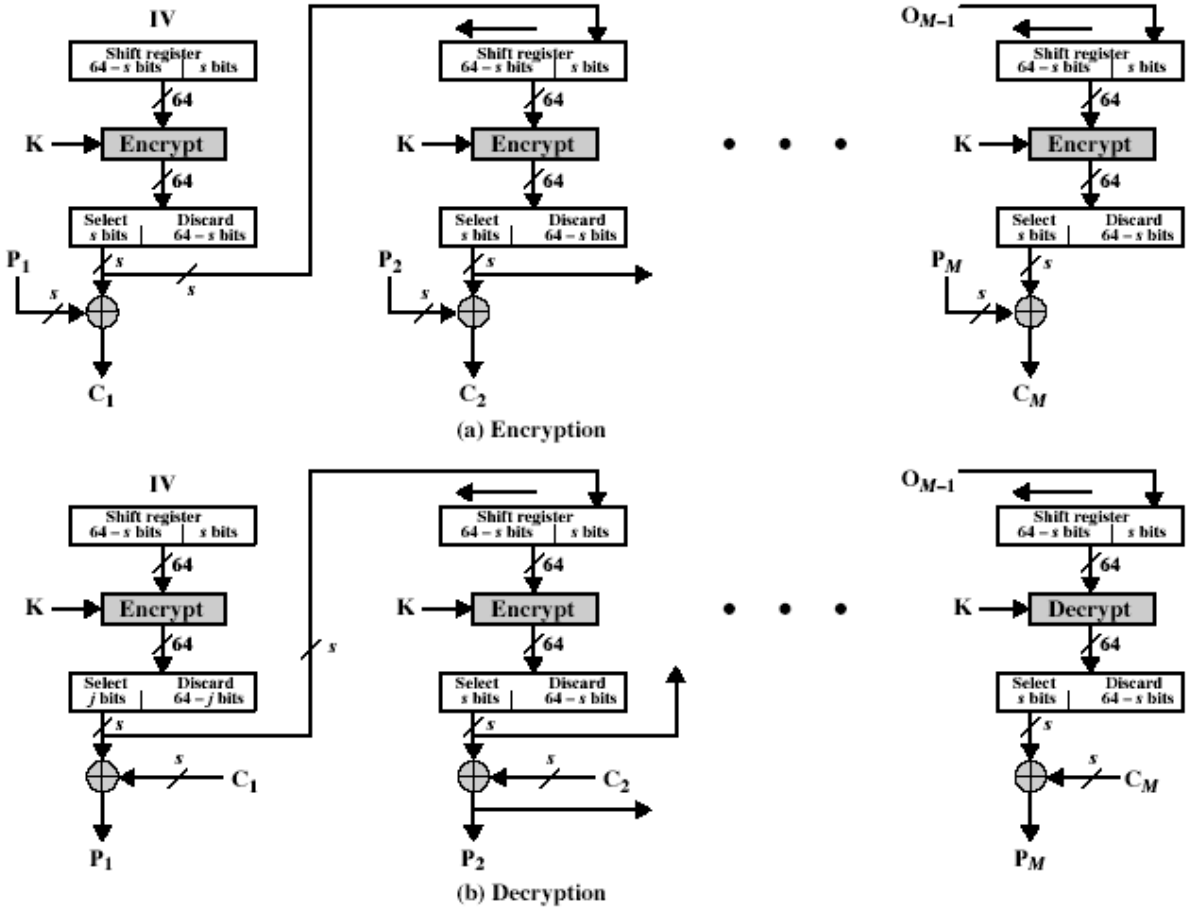


Advantages and Limitations of CFB

- 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

<u>OUTPUT FEEDBACK (OFB)</u>

- 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 = Pi XOR O_i
  $O_i = DES_{K1}(O_{i-1})$
  O_{-1} = IV
- uses: stream encryption over noisy channels

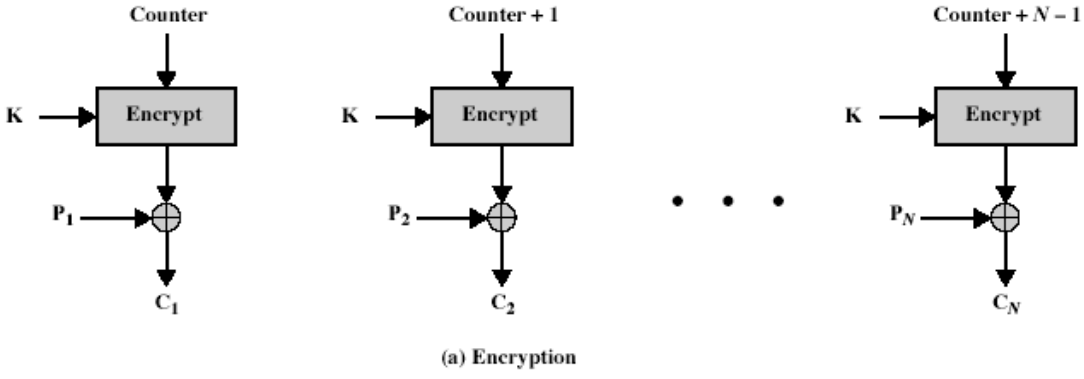


Advantages and Limitations of OFB

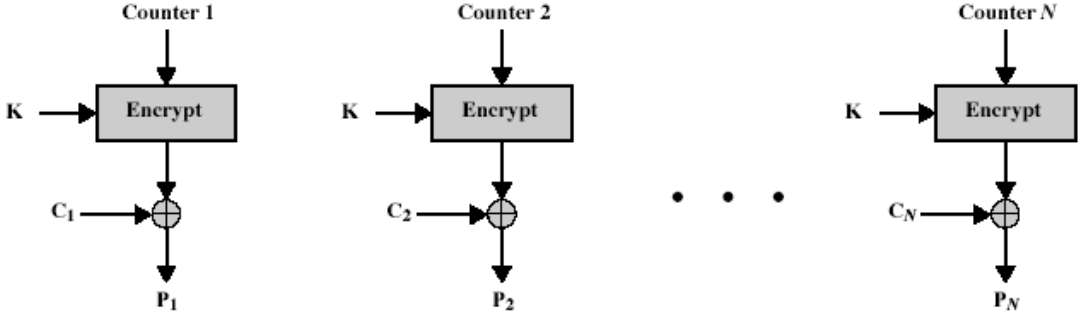
- used when error feedback a problem or where need to encryptions before message is available
- superficially similar to CFB
- but feedback is from the output of cipher and is independent of message
- a variation of a Vernam cipher
- hence must never reuse the same sequence (key+IV)
- sender and receiver must remain in sync, and some recovery method is needed to ensure this occurs
- originally specified with m-bit feedback in the standards
- subsequent research has shown that only **OFB-64** 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)
  $C_i = P_i$ XOR O_i
  $O_i = DES_{K1}(i)$
- uses: high-speed network encryptions



(a) Encryption



(b) Decryption

Advantages and Limitations of CTR

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

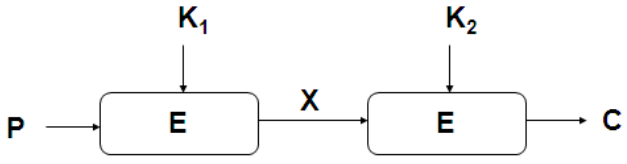
MULTIPLE ENCRYPTION & TRIPLEDES:

- Multiple encryption is a technique in which an encryption algorithm is used multiple times.
- Triple DES is one of the multiple encryption scheme using multiple keys..

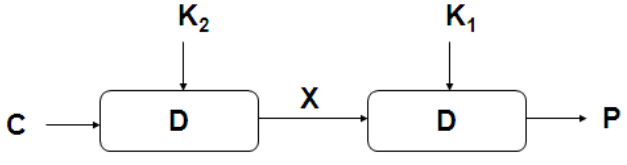
Double DES:

- Consits of two encryption stages.
- Uses two keys k1 and k2.

C= E(k2,E(k1,P).
P=D(k1,D(k2,C)).
It uses a keylength of 56 X 2=112 bits resulting an increase in cryptographic strength.



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



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

Disadvantages of Double DES:

- Reduction to a single stage.
- Meet in the middle attacks.
  - Meet-in-the-middle is a known attack that can exponentially reduce the number of brute force permutations required to decrypt text that has been encrypted by more than one key. Such an attack makes it much easier for an intruder to gain access to data.

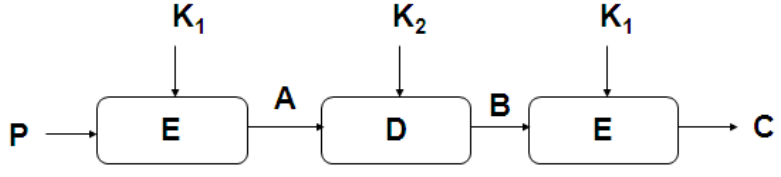
Triple-DES with Two-Keys:

- It has 3 encryptions
  - would seem to need 3 keys
- but can use 2 keys with E-D-E sequence
- Encryption

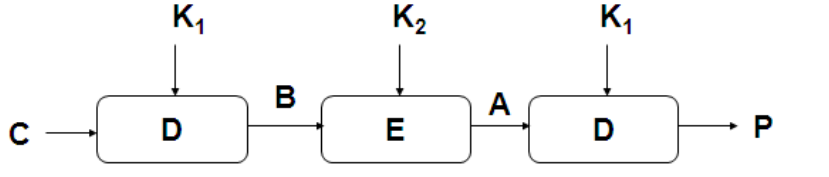
$C = E(K_1,D(K_2,E(K_1,P)))$

- Decryption

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



Encryption $C = E(K1,D(K2,E(K1,P)))$



Decryption $P = D(K1,E(K2,D(K1,C)))$

Triple-DES with Three-Keys:

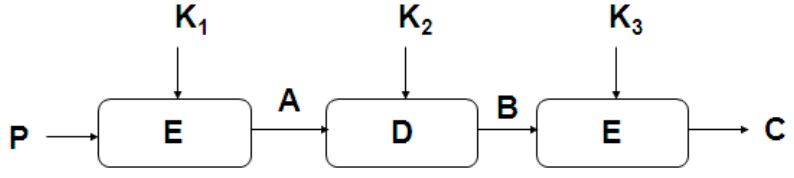
- Uses 3 keys with E-D-E sequence
- Encryption

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

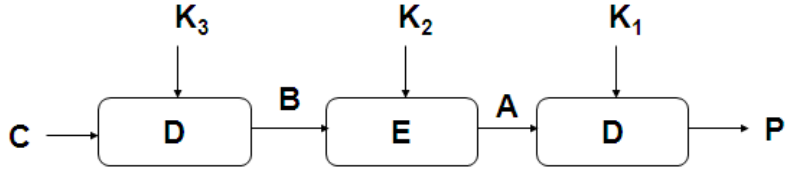
- Decryption

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

- has been adopted by some Internet applications, eg PGP, S/MIME



Encryption $C = E(K_3,D(K_2,E(K_1,P)))$



Decryption $P = D(K_1,E(K_2,D(K_3,C)))$

Advanced Encryption Standard (AES) :

AES Requirements

- private key symmetric block cipher
- 128-bit data, 128/192/256-bit keys
- stronger & faster than Triple-DES

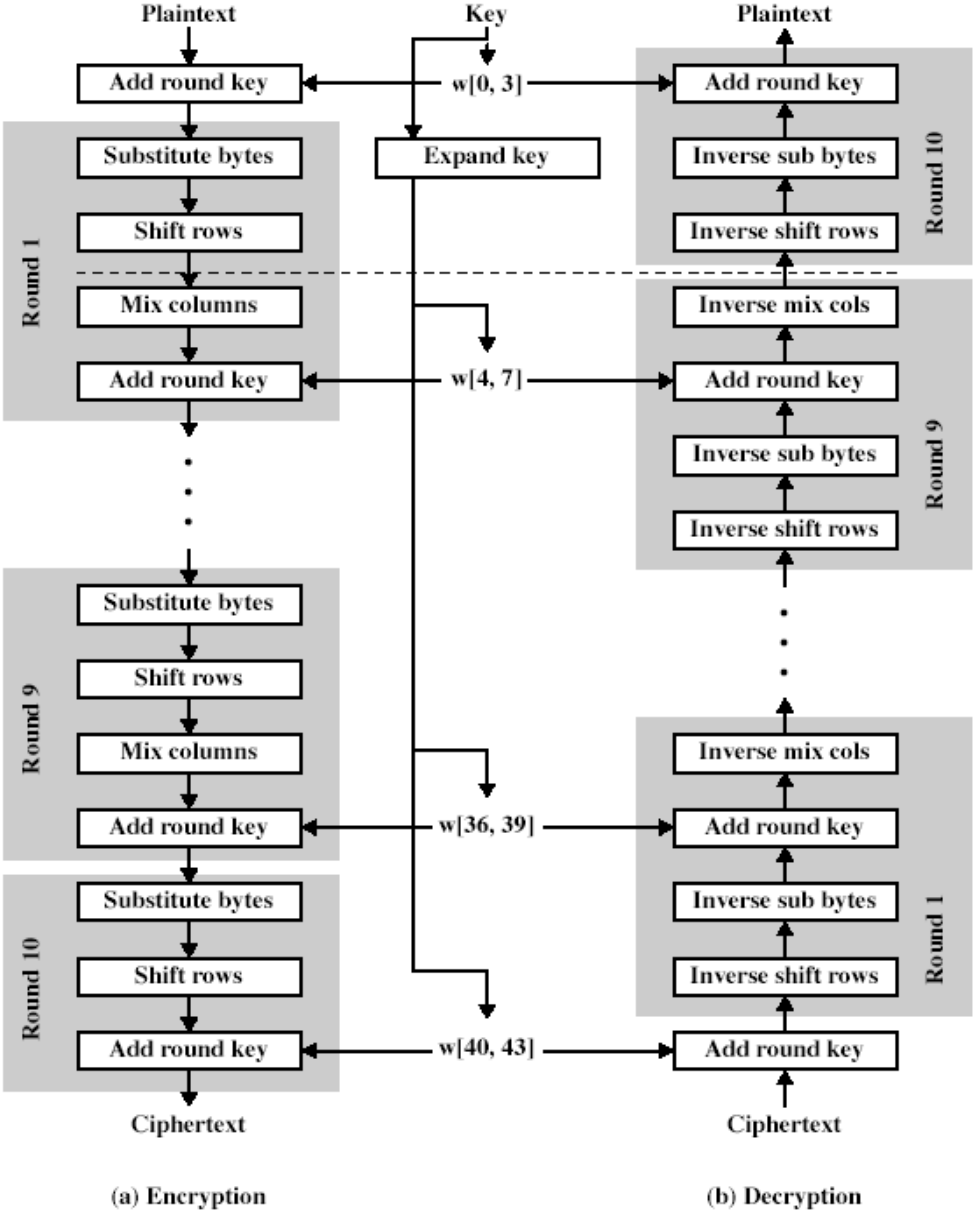
AES Evaluation Criteria

- initial criteria:
  - security – effort to practically cryptanalyse
  - cost – computational
  - algorithm & implementation characteristics
- final criteria
  - general security
  - software & hardware implementation ease
  - implementation attacks
  - flexibility (in en/decrypt, keying, other factors)

AES Cipher - Rijendael

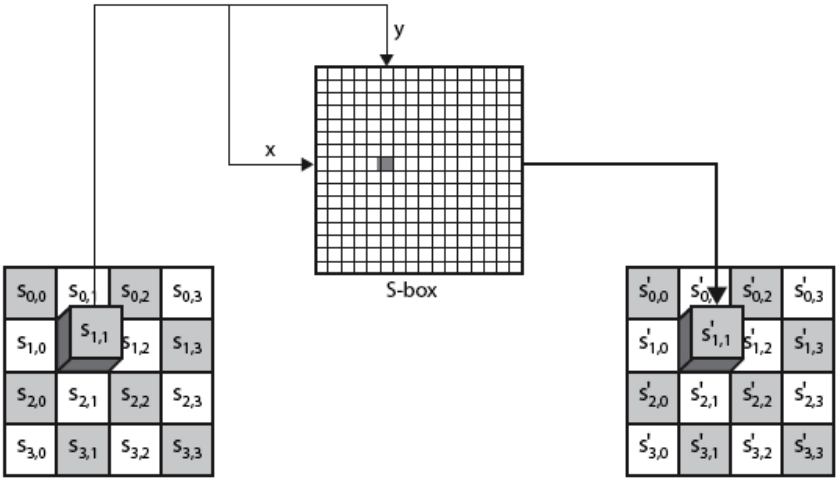
- designed by Rijmen-Daemen in Belgium
- has 128/192/256 bit keys, 128 bit data
- an iterative rather than feistel cipher
  - treats data in 4 groups of 4 bytes
  - operates an entire block in every round
- designed to be:
  - resistant against known attacks

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



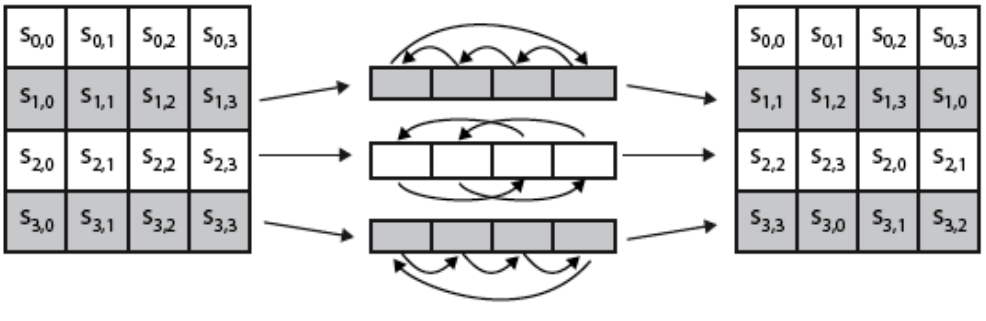
Byte Substitution

- a simple substitution of each byte
- uses one table of 16x16 bytes containing a permutation of all 256 8-bit values
- each byte of state is replaced by byte in row (left 4-bits) & column (right 4-bits)
- eg. byte {95} is replaced by row 9 col 5 byte
- which is the value {2A}
- S-box is constructed using a defined transformation of the values in GF(28)
- designed to be resistant to all known attacks



Shift Rows

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



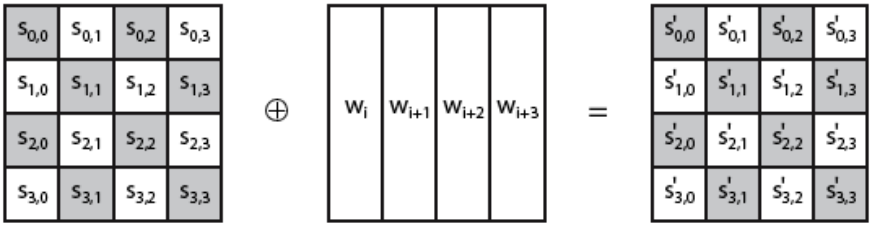
Mix Columns

- each column is processed separately
- each byte is replaced by a value dependent on all 4 bytes in the column
- effectively a matrix multiplication in GF(28) using prime poly m(x) =x8+x4+x3+x+1

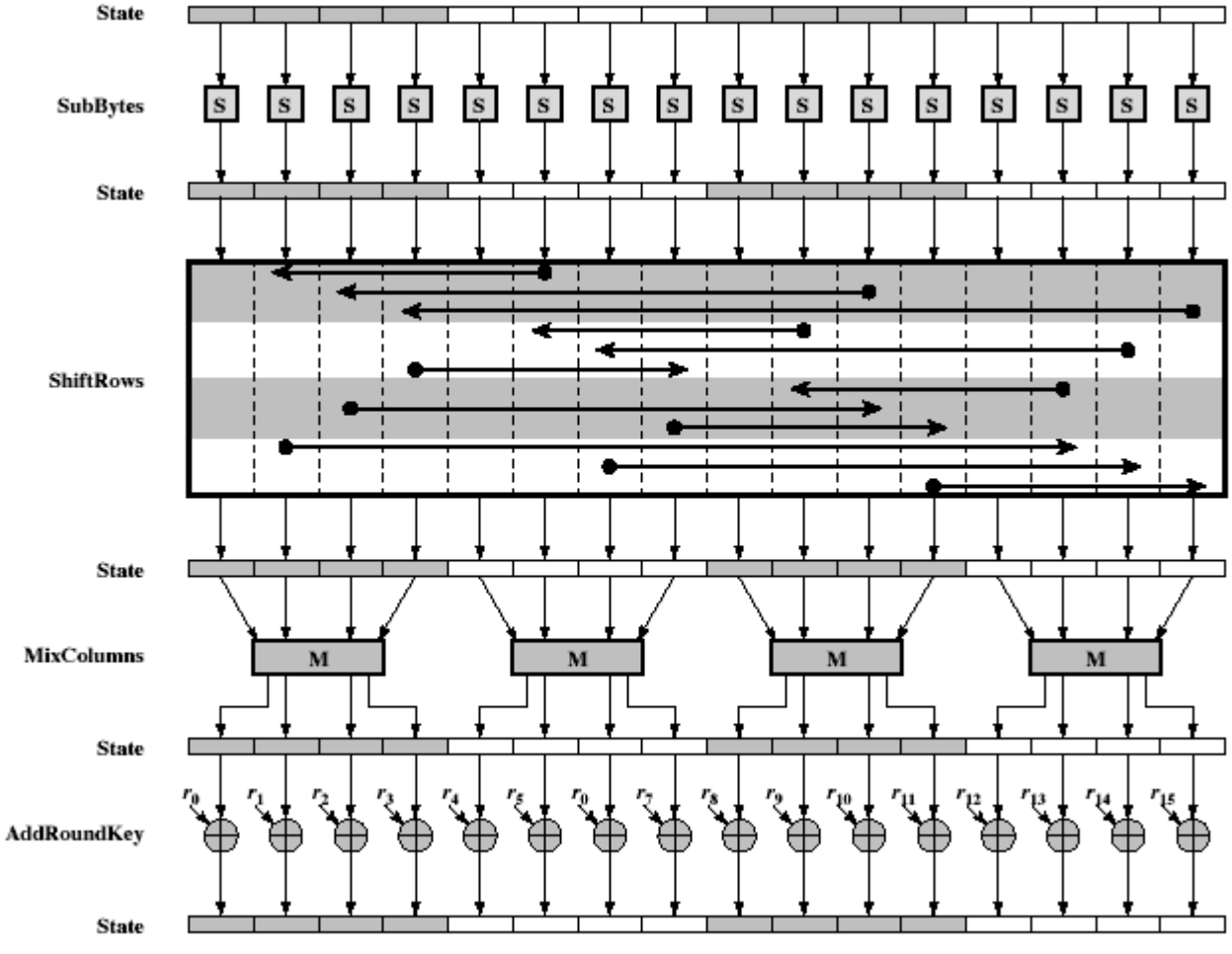
$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix}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

Add Round Key

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

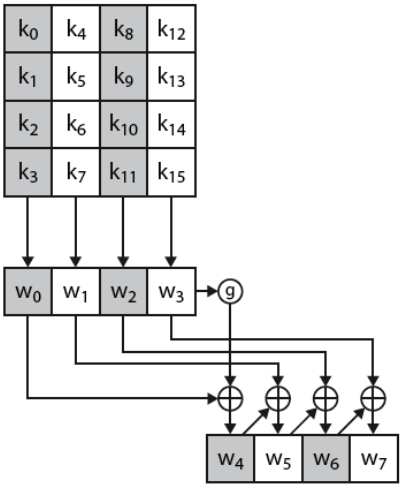


AES Round:



AES Key Expansion

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



AES Decryption

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

PUBLIC-KEY CRYPTOGRAPHY AND RSA:

Private-Key Cryptography

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

Public-Key Cryptography

- probably most significant advance in the 3000 year history of cryptography
- uses two keys – a public & a private key
- asymmetric since parties are not equal
- uses clever application of number theoretic concepts to function
- complements rather than replaces private key crypto
- public-key/two-key/asymmetric cryptography involves the use of two keys:

- a public-key, which may be known by anybody, and can be used to encrypt messages, and verify signatures
- a private-key, known only to the recipient, used to decrypt messages, and sign (create) signatures

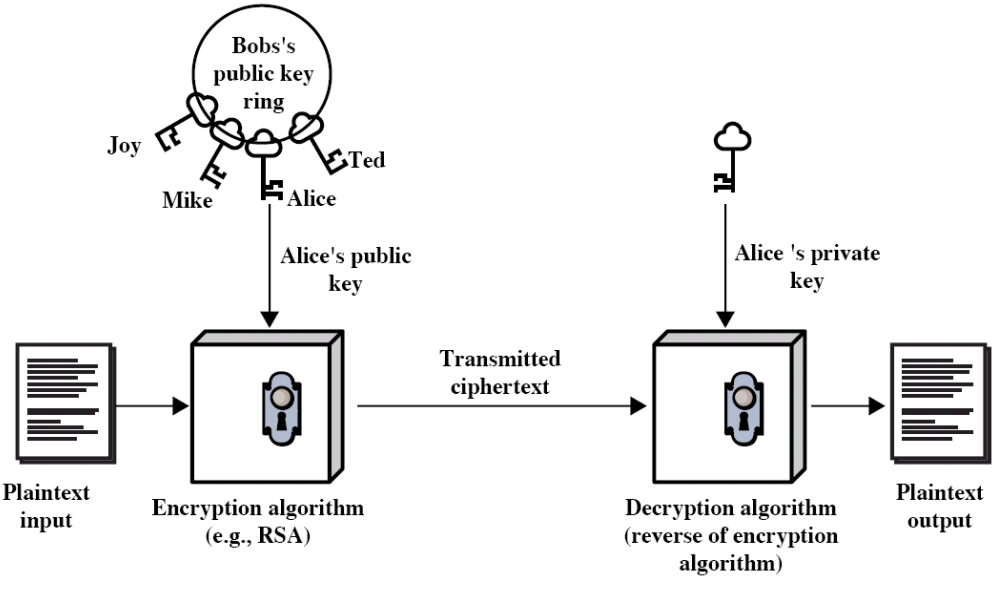
- is asymmetric because
  - those who encrypt messages or verify signatures cannot decrypt messages or create signatures

Why Public-Key Cryptography?

- developed to address two key issues:
  - key distribution – how to have secure communications in general without having to trust a KDC with your key
  - digital signatures – how to verify a message comes intact from the claimed sender
- public invention due to Whitfield Diffie & Martin Hellman at Stanford University in 1976
  - known earlier in classified community

Public-Key Characteristics

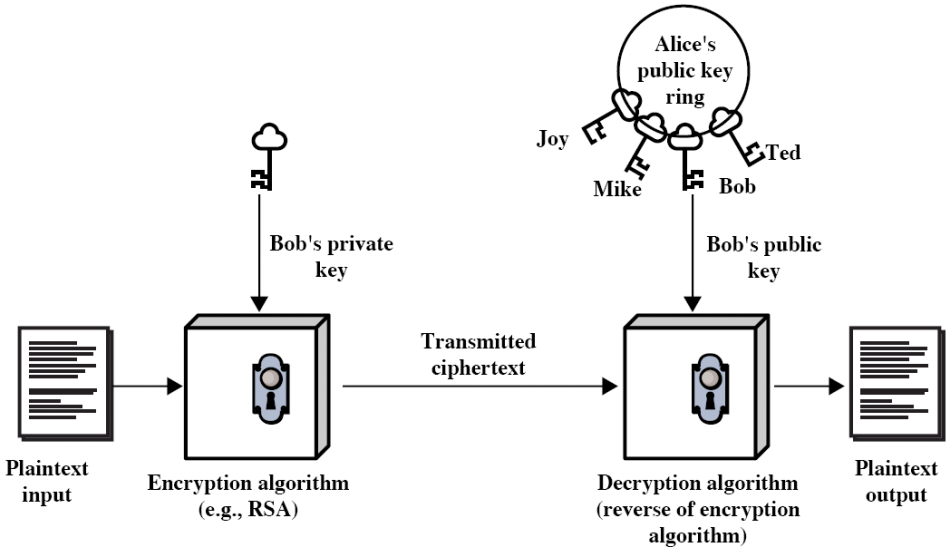
- Public-Key algorithms rely on two keys with the characteristics that it is:
  - computationally infeasible to find decryption key knowing only algorithm & encryption key
  - computationally easy to en/decrypt messages when the relevant (en/decrypt) key is known
  - either of the two related keys can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



(a) Encryption

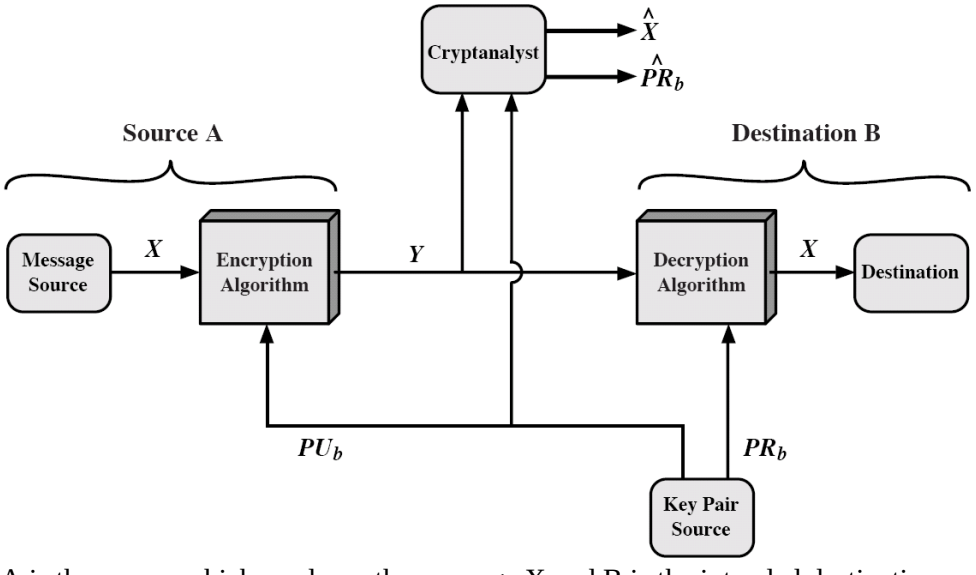
Difference between Conventional and Public-Key Encryption:

Conventional (Symmetric)	Public-Key (Asymmetric)
1. Same algorithm and key used for encryption and decryption	1. Same algorithm but different keys used for encryption and decryption
2. Parties share algorithm and key	2. Parties share algorithm but each has one key from a matched pair
3. Key must be kept secret	3. One key must be kept secret
4. Cipher must be strong	4. Cipher must be strong



(b) Authentication

Public-Key Cryptography – Secrecy:

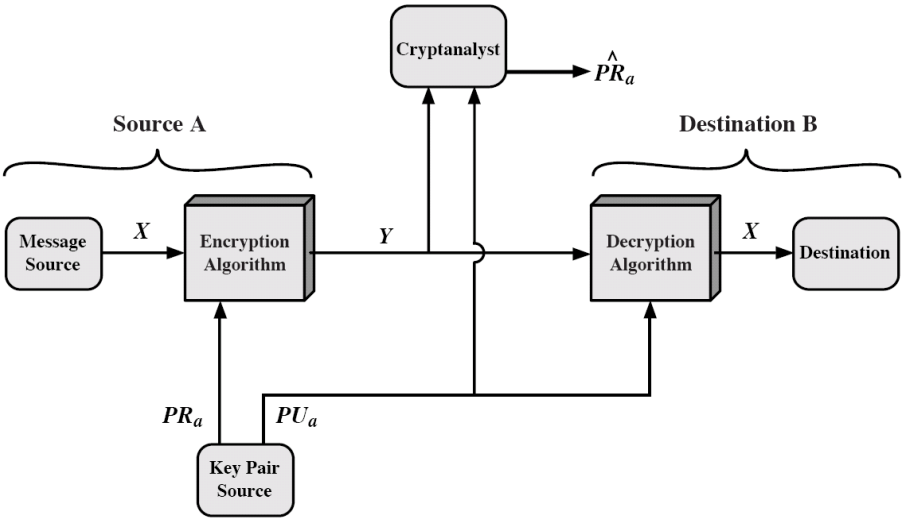


- Here A is the source which produces the message X and B is the intended destination.
- B generates a pair of keys
- PR_b is known to B only and PU_b is publicly available. so that it is accessible by A
- With X and PU_b as input A forms the cipher text.

  $Y = EPU_b(X)$
- The receiver then uses the private key PR_b to decrypt the message.

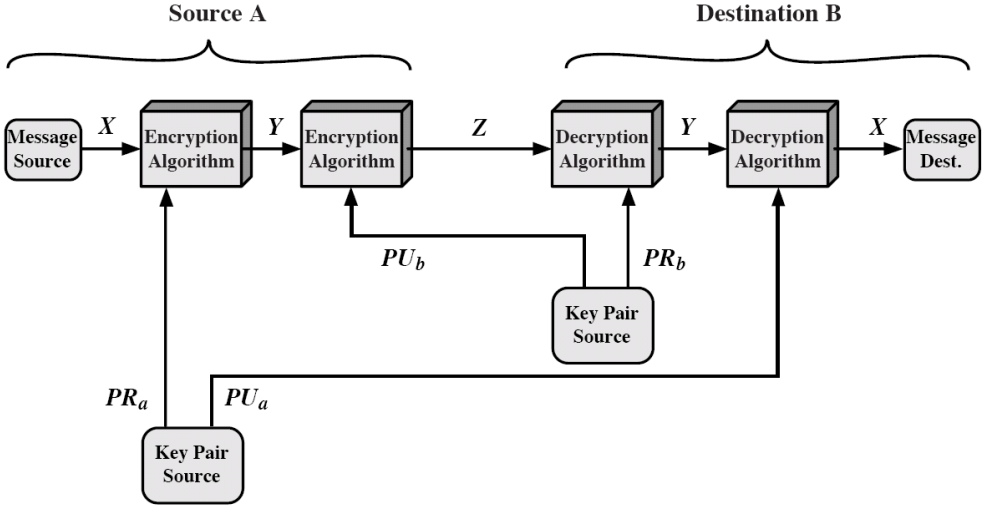
  $X = DPR_b(Y)$

PKC: Authentication



- Here **A** is the source which produces the message **X** and **B** is the intended destination
- **A** generates a pair of keys
  - $\mathbf{PU_a}$: A's public key
  - $\mathbf{PR_a}$: A's private key
- **A** encrypt the message using its private key **PRa** and transmit it to **B.**
- **B** decrypt the message using **A's** Public key.
- Here the encrypted message act as a Digital signature.

PKC: Secrecy and Authentication



- It also possible to provide both authentication as well as secrecy by a double use of Public Key scheme
  - $Z = E_{PUb}[E_{PRa}(X)]$

- $X = D_{PUa}[D_{PRb}(Z)]$

- Here the sender first encrypt the message using the sender's private key, it acts as a digital signature
- Then the sender again encrypt the message using receiver's Public key, which provides secrecy

RSA Algorithm:

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

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

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

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

RSA Key Setup

- each user generates a public/private key pair by:
- selecting two large primes at random - p, q

- computing their system modulus N=p.q
    - note ø(N)=(p-1)(q-1)
- selecting at random the encryption key e
    - where 1<e<ø(N), gcd(e,ø(N))=1
- solve following equation to find decryption key d
    - e.d=1 mod ø(N) and 0≤d≤N
- publish their public encryption key: KU={e,N}
- keep secret private decryption key: KR={d,N}

RSA Use

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

RSA Example

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

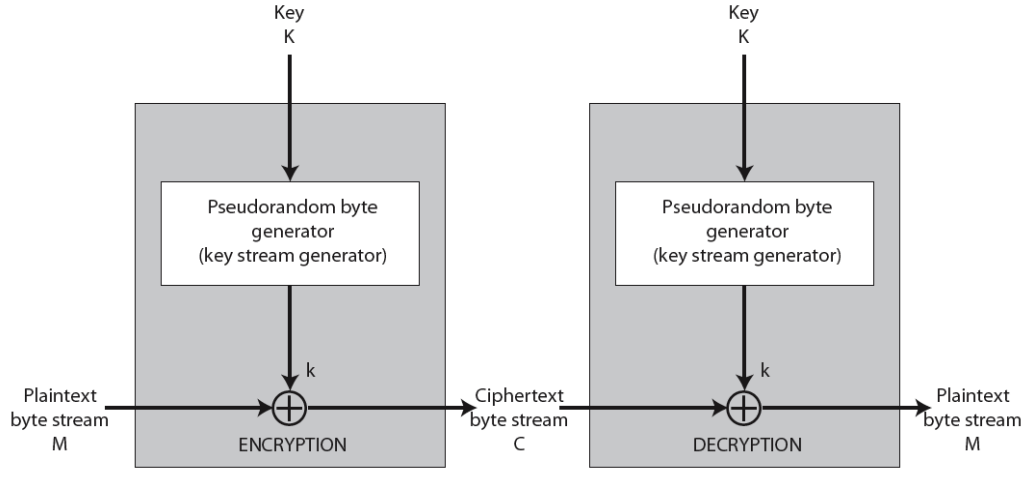
sample RSA encryption/decryption is:
given message M = 88 (nb. 88<187)
encryption:
C = 887 mod 187 = 11
decryption:
M = 1123 mod 187 = 88

Attacks on RSA Algorithm:

- Brute force (Key space search)
    - Try all possible private keys
    - Use large keys
- Mathematical Attacks
    - Several approaches, all equivalent to factoring
- Timing attacks
    - Based on the running time of the decryption algorithm

STREAM CIPHERS:

- A stream cipher is a symmetric encryption algorithm in which ciphertext output is produced bit by bit or byte by byte from a stream of plaintext input.
- Most commonly used cipher is RC4.



- A key is input to a pseudorandom bit generator that produce a stream of 8 bit numbers that are random.
- Output of the generator called a key stream.

Encrtyption:

Plaintext: 11001100

Key stream: 01101100

Ciphertext: 10100000

DECRYPTION:

Plaintext: 11001100

Design considerations for stream cipher:

- The encryption sequence should have a large period.
- The key stream should approximate the properties of a true random number stream as close as possible.
- The key should be sufficiently long to protect from brute force attack.

Advantages:

1. Faster
2. Use less code than block cipher.

Applications:

1. Used in data communication channel.

RC4:

- RC4 is a stream cipher designed in 1987 by ron Rivest for RSA security.
- It is a variable key size stream cipher with byte oriented operations.
- Algorithm is based on random permutation.
- RC4 is used in SSL/TLS (Transport Layer Security protocol, Secure Sockets Layer protocol) standards in web browsers and servers.

Working principle:

- A variable length key of from 1 to 256 bytes is used to initialize a 256 byte state vector S, with elements S[0],S[1],....S[255].
- S contains a permutation of all 8 bit number from 0 through 255.
- A byte k is generated by S by selecting one of the 255 entries in a systematic fashion.

Description:

- Two main parts:
  - KSA (Key Scheduling Algorithm)
  - PRGA (Pseudo Random Generation Algorithm)
- Notation:
  - S = {0, 1, 2, … N-1} is the initial permutation
  - L(keylen) = length of key

Initialization of S

- starts with an array S of numbers: 0..255
- Temporary vector T is created.
- If the length of key is 256 bytes then K is transferred to T.
- Else the the keylength of T are copied from K and then k is repeated as many times as necessary to fill out T.

for i = 0 to 255 do

S[i] = i

T[i] = K[i mod keylen])

Initial permutation of S

- Starting with s[0] through s[255], for each s[i] swapping s[i] with another byte in S.

j = 0

for i = 0 to 255 do

j = (j + S[i] + T[i]) (mod 256)

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

Stream generation(PRGA):

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]

Encryption: XOR the value k with the next byte of plaintext.

Decryption: XOR the value k with the next byte of cipher text.

Example:

- Simple 4-byte example
- S = {0, 1, 2, 3}
- K = {1, 7, 1, 7}
- Set i = j = 0

KSA(key schedule algorithm)

First Iteration (i = 0, j = 0, S = {0, 1, 2, 3}):

j = (j + S[i] + K[i]) = (0 + 0 + 1) = 1

Swap S[i] with S[j]: S = {1, 0, 2, 3}

Second Iteration (i = 1, j = 1, S = {1, 0, 2, 3}):

j = (j + S[i] + K[i]) = (1 + 0 + 7) = 0 (mod 4)

Swap S[i] with S[j]: S = {0, 1, 2, 3}

Third Iteration (i = 2, j = 0, S = {0, 1, 2, 3}):

j = (j + S[i] + K[i]) = (0 + 2 + 1) = 3

Swap S[i] with S[j]: S = {0, 1, 3, 2}

Fourth Iteration (i = 3, j = 3, S = {0, 1, 3, 2}):

j = (j + S[i] + K[i]) = (3 + 2 + 7) = 0 (mod 4)

Swap S[i] with S[j]: S = {2, 1, 3, 0}

PRGA (pseudo random byte generator)

Reset i = j = 0, Recall S = {2, 1, 3, 0}

i = i + 1 = 1

j = j + S[i] = 0 + 1 = 1

Swap S[i] and S[j]: S = {2, 1, 3, 0}

T= S[i] + S[j] = 2

Output K = S[T] = S[2] = 3

PRIMALITY TESTING:

- The primality test provides the probability of whether or not a large number is prime.
- Several theorems including Fermat's theorem provide idea of primality test.
- Cryptography schemes such as RSA algorithm heavily based on primality test.
- A Prime number is an integer that has no integer factors other than 1 and itself. On the other hand, it is called composite number.
- A primality testing is a test to determine whether or not a given number is prime, as opposed to actually decomposing the number into its constituent prime factors (which is known as prime factorization) Use multiple points if necessary.

Algorithms:

- **A Naïve Algorithm**
  - Pick any integer P that is greater than 2.
  - Try to divide P by all odd integers starting from 3 to square root of P.
  - If P is divisible by any one of these odd integers, we can conclude that P is composite.

- The worst case is that we have to go through all odd number testing cases up to square root of P.
- Time complexity is O(square root of N)

- **Fermat's Theorem**
  - Given that P is an integer that we would like to test that it is either a PRIME or not.
  - And A is another integer that is greater than zero and less than P.
  - From Fermat's Theorem, if P is a PRIME, it will satisfy this two equalities:
    - A^(p-1) = 1(mod P) or A^(p-1)mod P = 1
    - A^P = A(mod P) or A^P mod P = A
  - For instances, if P = 341, will P be PRIME?
  - from previous equalities, we would be able to obtain that:
    - 2^(341-1)mod 341 = 1, if A = 2
  - It seems that 341 is a prime number under Fermat's Theorem. However, if A is now equal to 3:
  - 3^(341-1)mod 341 = 56 !!!!!!!!!
  - That means Fermat's Theorem is not true in this case!
  - $\Rightarrow$ Time complexity is O(log n) $\Leftarrow$

- **Rabin-Miller's Probabilistic Primality Algorithm**
  - The Rabin-Miller's Probabilistic Primality test was by Rabin, based on Miller's idea. This algorithm provides a fast method of determining of primality of a number with a controllably small probability of error.
  - Given (b, n), where n is the number to be tested for primality, and b is randomly chosen in [1, n-1]. Let n-1 = (2^q)*m, where m is an odd integer.
    - B^m = 1(mod n)
    - $\exists i \in$[0, q-1] such that b^(m2)^i = -1(mod n)
  - If the testing number satisfies either cases, it will be said as "inconclusive". That means it could be a prime number.
  - From Fermat's Theorem, it concludes 341 is a prime but it is 11 * 31!
  - Now try to use Rabin-Miller's Algorithm.
  - Let n be 341, b be 2. then assume:

- q = 2 and m = 85 (since, n -1 = 2^q*m)
- 2^85 mod 341 = 32
- Since it is not equal to 1, 341 is composite!
- Time complexity is O(log N)

- **Solovay–Strassen primality test**
  - Given an odd number *n* we can contemplate whether or not the congruence

$$a^{(n-1)/2} \equiv \left(\frac{a}{n}\right) \pmod{n}$$

  - holds for various values of the "base" *a*. If *n* is prime then this congruence is true for all *a*.
  -