CSEN1001

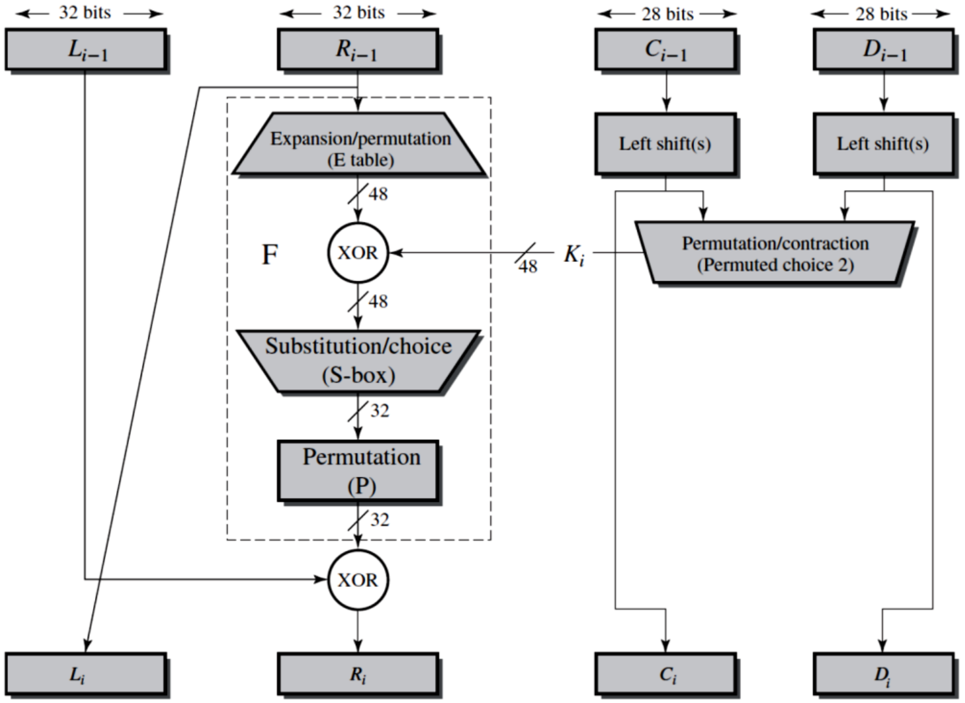
Computer and Network Security

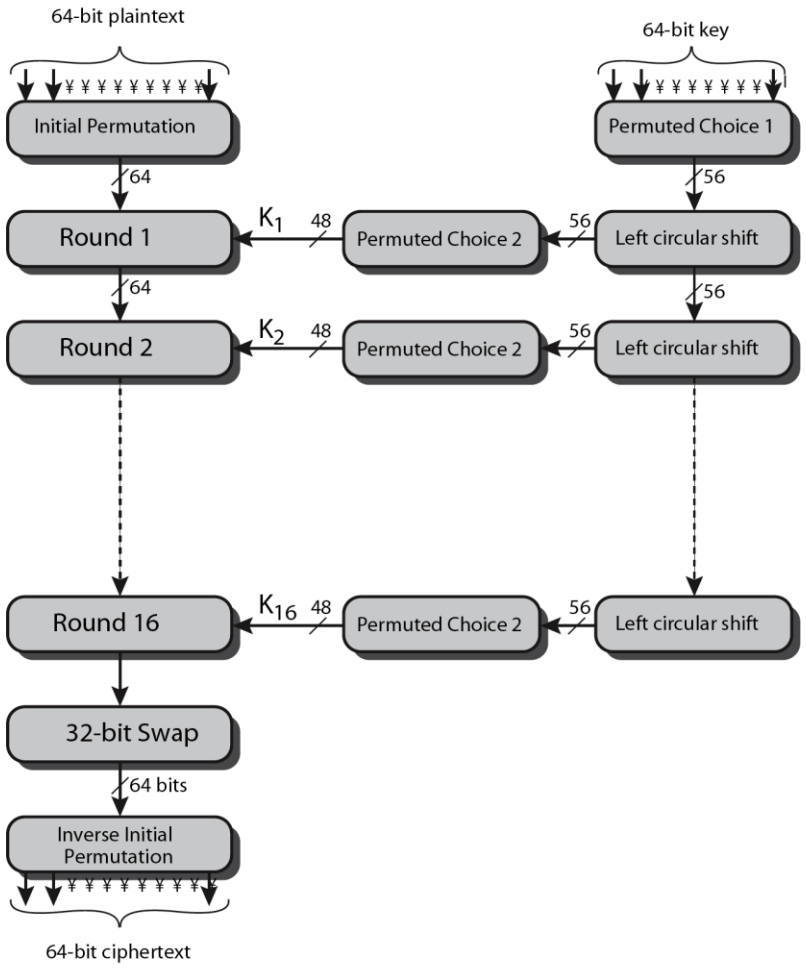
Mervat AbuElkheir
Mohamed Abdelrazik
Ahmad Helmy

Lecture (4)

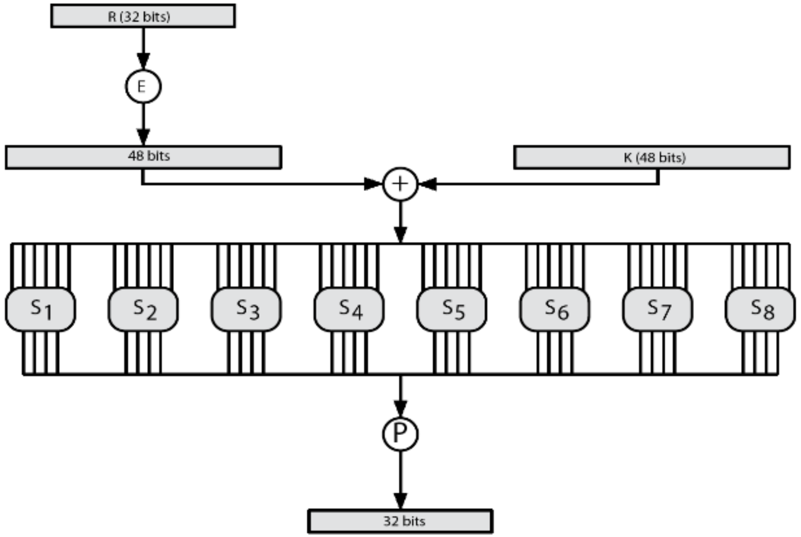
Block Ciphers – cont’d

DES Encryption





DES Encryption



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

Why DES works the way it does

"Differential cryptanalysis [5] is based on the fact that in many s-boxes certain input XORs (i.e., certain fixed changes in the s-box input vector) lead to certain output XORs (fixed changes in the s-box output vector) with fairly high probability ([about] 25%) and to certain other output XORs with very low or zero probability. Chosen plaintext attacks can be mounted which take advantage of the relatively high probabilities to reduce the search space for the key in use. It is obvious, therefore, that if all output XORs occurred with similar (ideally, equal) probability, differential cryptanalysis would have no greater chance of success than exhaustive search."

[Adams, C. 1992. On immunity against Biham and Shamir's "differential cryptanalysis." *Information Processing Letters.* 41: 77-80.]

Why DES works the way it does

". . . the resulting cryptogram exhibits a sensitive intersymbol dependence that makes all output digits complicated functions not only of all message digits but also of all digits in the key."

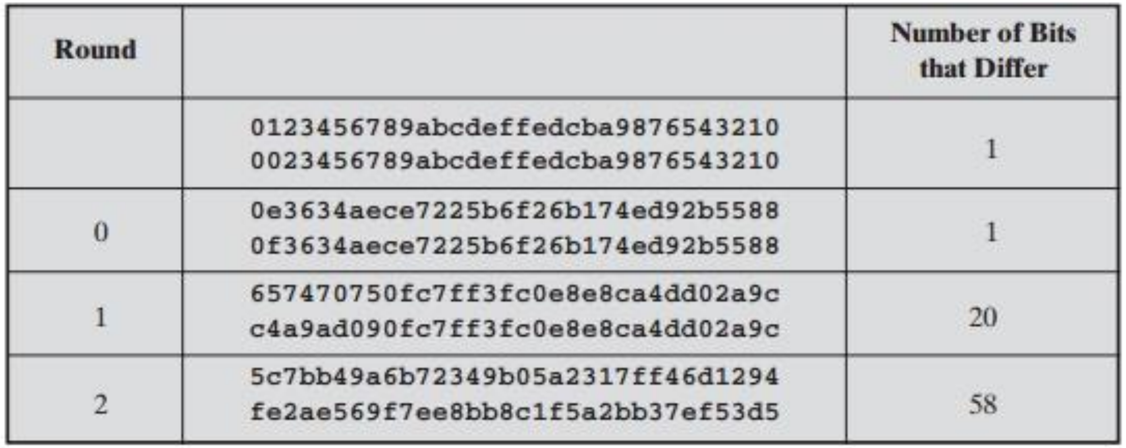
[Feistel, H. 1973. Cryptography and Computer Privacy. *Scientific American.* 228(5): 15-23]

For a complete list of papers that provided mathematical analysis of how the many substitutions in block ciphers achieve confusion and diffusion:

http://www.ciphersbyritter.com/RES/SBOXDESN.HTM#Feistel73

Avalanche Effect

- key desirable property of encryption algorithm
- 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



Round		Number of Bits that Differ
	0123456789abcdeffedcba9876543210 0023456789abcdeffedcba9876543210	1
0	0e3634aece7225b6f26b174ed92b5588 0f3634aece7225b6f26b174ed92b5588	1
1	657470750fc7ff3fc0e8e8ca4dd02a9c c4a9ad090fc7ff3fc0e8e8ca4dd02a9c	20
2	5c7bb49a6b72349b05a2317ff46d1294 fe2ae569f7ee8bb8c1f5a2bb37ef53d5	58

Change in Plaintext		Change in Key	
Round	Bit diff	Round	Bit diff
0	1	0	0
1	6	1	2
2	21	2	14
3	35	3	28
4	39	4	32
5	34	5	30
6	32	6	32
7	31	7	35
8	29	8	34
9	42	9	40
10	44	10	38
11	32	11	31
12	30	12	33
13	30	13	28
14	26	14	26
15	29	15	34
16	34	16	35

DES Decryption

- Decryption 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

DES Strength – 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

Advanced Encryption Standard (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
- ❑ provide full specification & design details
- ❑ both C & Java implementations
- ❑ NIST have released all submissions & unclassified analyses

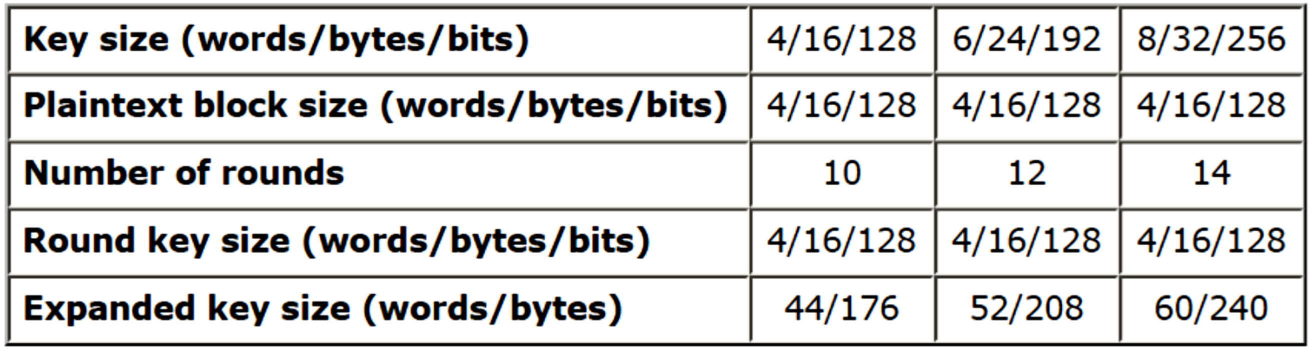
Evaluation Criteria

- ➢ 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)

Advanced Encryption Standard (AES) - Rijndael

- ❑ designed by Rijmen-Daemen in Belgium
- ❑ an **iterative** rather than **Feistel** cipher
  - processes data as block of 4 columns of 4 bytes
  - operates on entire data block in every round

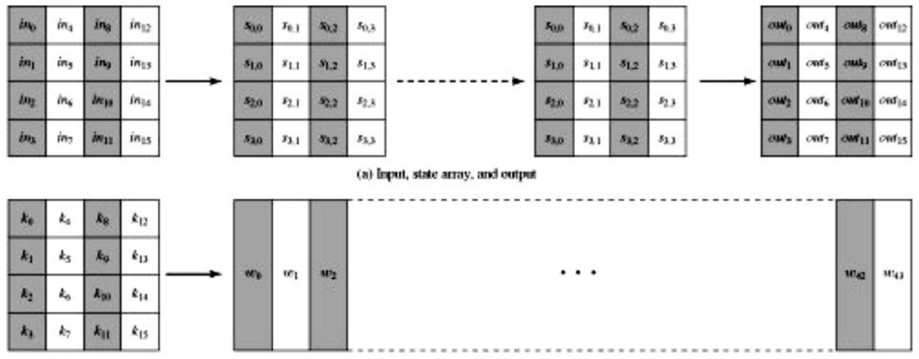
- ❑ designed to have:
  - resistance against known attacks
  - speed and code compactness on many CPUs
  - design simplicity



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

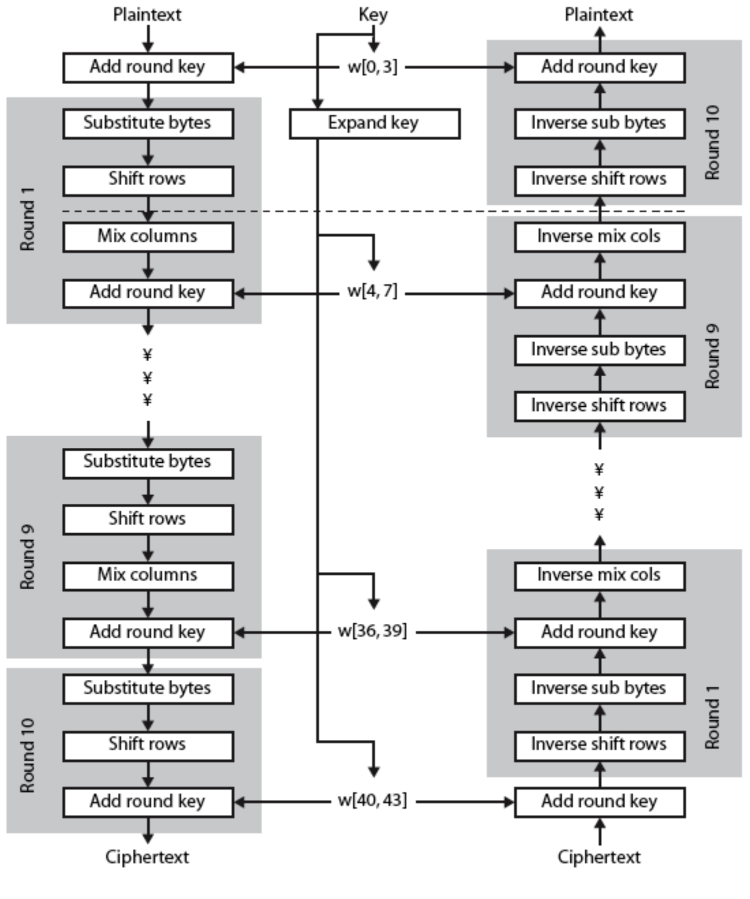
Advanced Encryption Standard (AES) - Rijndael

- 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 multiple of groups)
  - add round key (XOR state with key material)
- initial XOR key material & incomplete last round
- with fast XOR & table lookup implementation



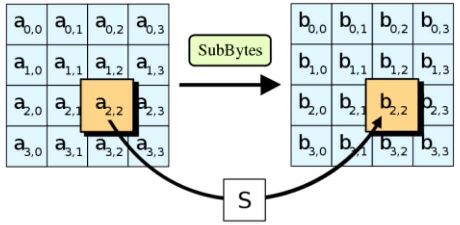
(a) Input, state array, and output

(b) Key and expanded key



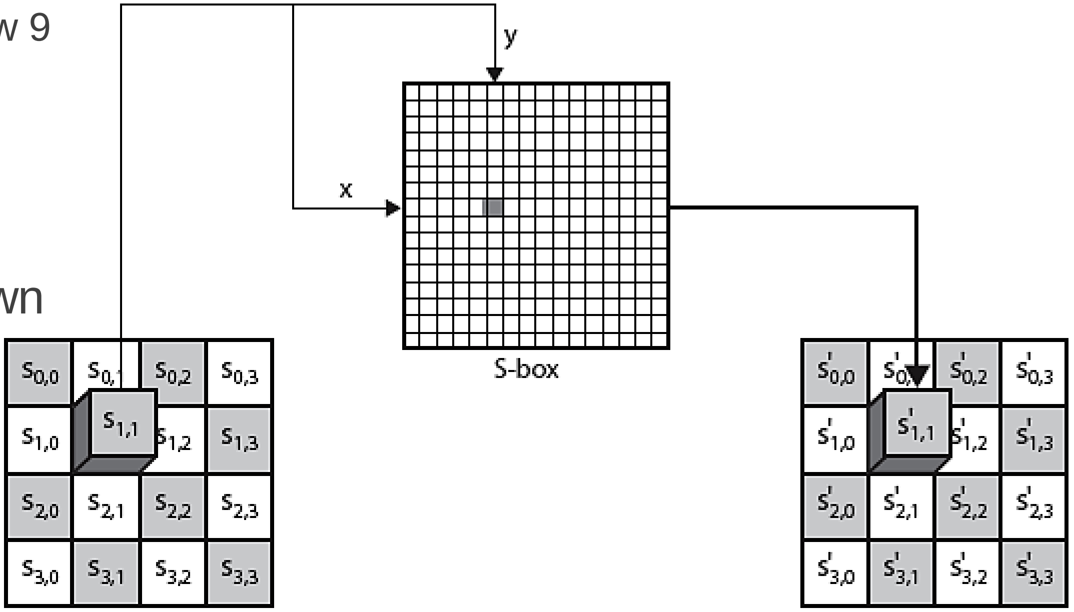
(a) Encryption (b) Decryption

Byte Substitution



$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
SubBytes
S
$b_{0,0}$ $b_{0,1}$ $b_{0,2}$ $b_{0,3}$
$b_{1,0}$ $b_{1,1}$ $b_{1,2}$ $b_{1,3}$
$b_{2,0}$ $b_{2,1}$ $b_{2,2}$ $b_{2,3}$
$b_{3,0}$ $b_{3,1}$ $b_{3,2}$ $b_{3,3}$

- 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 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
- designed to be resistant to all known attacks



y
x
S-box
$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,0}$ $s_{1,1}$ $s_{1,2}$ $s_{1,3}$
$s_{2,0}$ $s_{2,1}$ $s_{2,2}$ $s_{2,3}$
$s_{3,0}$ $s_{3,1}$ $s_{3,2}$ $s_{3,3}$
$s'_{0,0}$ $s'_{0,1}$ $s'_{0,2}$ $s'_{0,3}$
$s'_{1,0}$ $s'_{1,1}$ $s'_{1,2}$ $s'_{1,3}$
$s'_{2,0}$ $s'_{2,1}$ $s'_{2,2}$ $s'_{2,3}$
$s'_{3,0}$ $s'_{3,1}$ $s'_{3,2}$ $s'_{3,3}$

(a) S-box

y

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

(b) Inverse S-box

y

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
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

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

Number	0	1	2	3	4	5	6	7
Binary	0000	0001	0010	0011	0100	0101	0110	0111
Hexadecimal	0	1	2	3	4	5	6	7

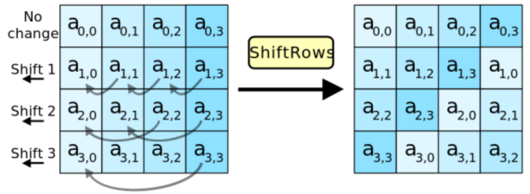
Number	8	9	10	11	12	13	14	15
Binary	1000	1001	1010	1011	1100	1101	1110	1111
Hexadecimal	8	9	A	B	C	D	E	F

Circuit Globe

Byte Substitution - Rationale

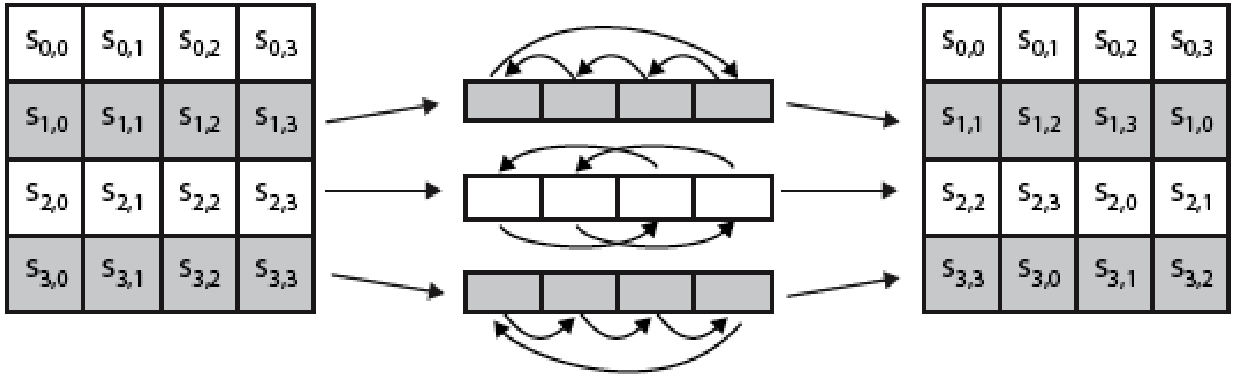
- ❑ Resistant to cryptanalysis attacks
- ❑ Low correlation between input bits and output bits
- ❑ No fixed points and no opposite fixed points $[Sbox(a) = \bar{a}]$, where $\bar{a}$ is the bitwise complement of a
- ❑ Must be invertible, but the S-box is not self-inverse $[Sbox\{95\} = \{2A\}]$ but $[ISbox\{95\} = \{AD\}]$

Shift Rows



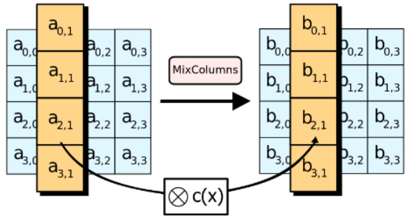
No change
Shift 1
Shift 2
Shift 3
ShiftRows
$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,1}$ $a_{1,2}$ $a_{1,3}$ $a_{1,0}$
$a_{2,2}$ $a_{2,3}$ $a_{2,0}$ $a_{2,1}$
$a_{3,3}$ $a_{3,0}$ $a_{3,1}$ $a_{3,2}$

- a circular byte shift
  - 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



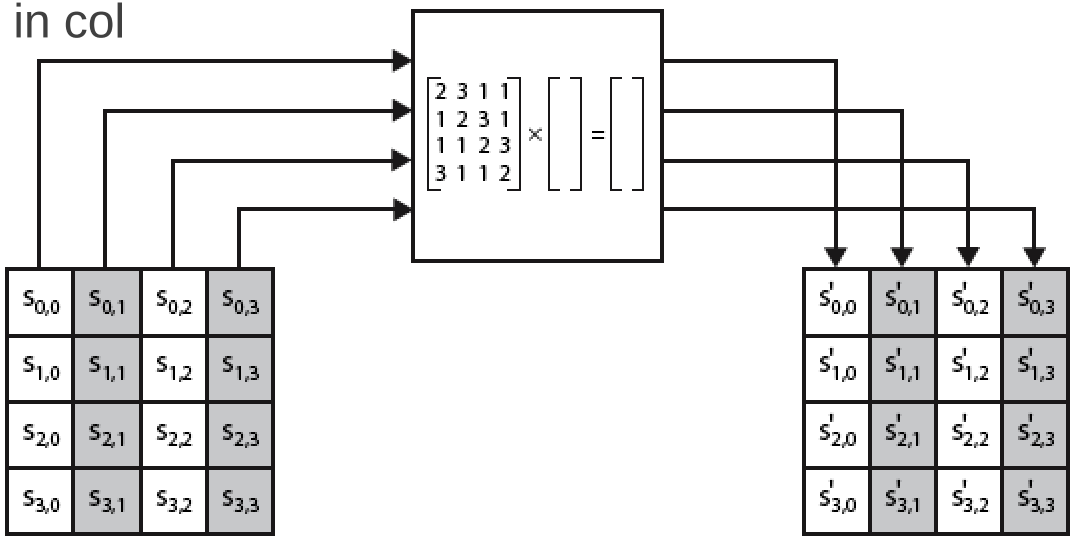
$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,0}$ $s_{1,1}$ $s_{1,2}$ $s_{1,3}$
$s_{2,0}$ $s_{2,1}$ $s_{2,2}$ $s_{2,3}$
$s_{3,0}$ $s_{3,1}$ $s_{3,2}$ $s_{3,3}$
$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,1}$ $s_{1,2}$ $s_{1,3}$ $s_{1,0}$
$s_{2,2}$ $s_{2,3}$ $s_{2,0}$ $s_{2,1}$
$s_{3,3}$ $s_{3,0}$ $s_{3,1}$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

Mix Columns



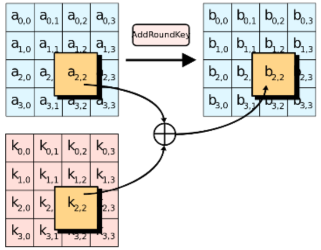
- each column is processed separately
- each byte is replaced by a value dependent on all 4 bytes in the column – effectively a matrix multiplication
- can express each col as 4 equations
  - to derive each new byte in col

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

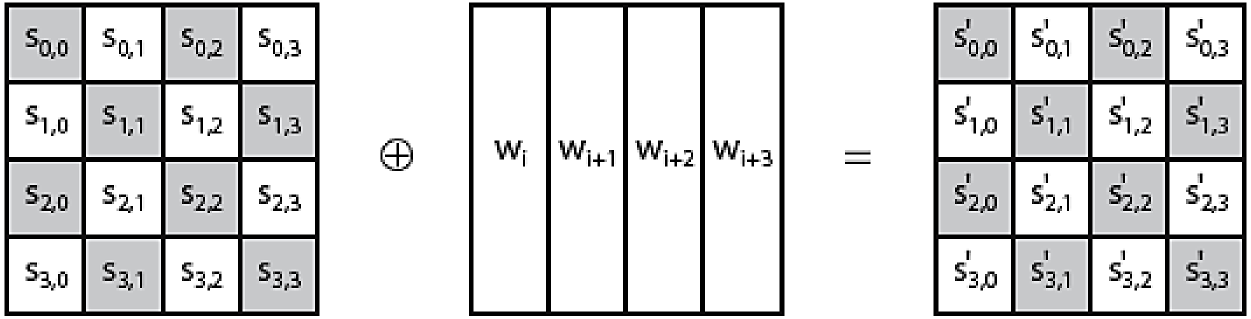


$$\begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \times \begin{bmatrix} \\ \\ \\ \end{bmatrix} = \begin{bmatrix} \\ \\ \\ \end{bmatrix}$$

Add Round Key

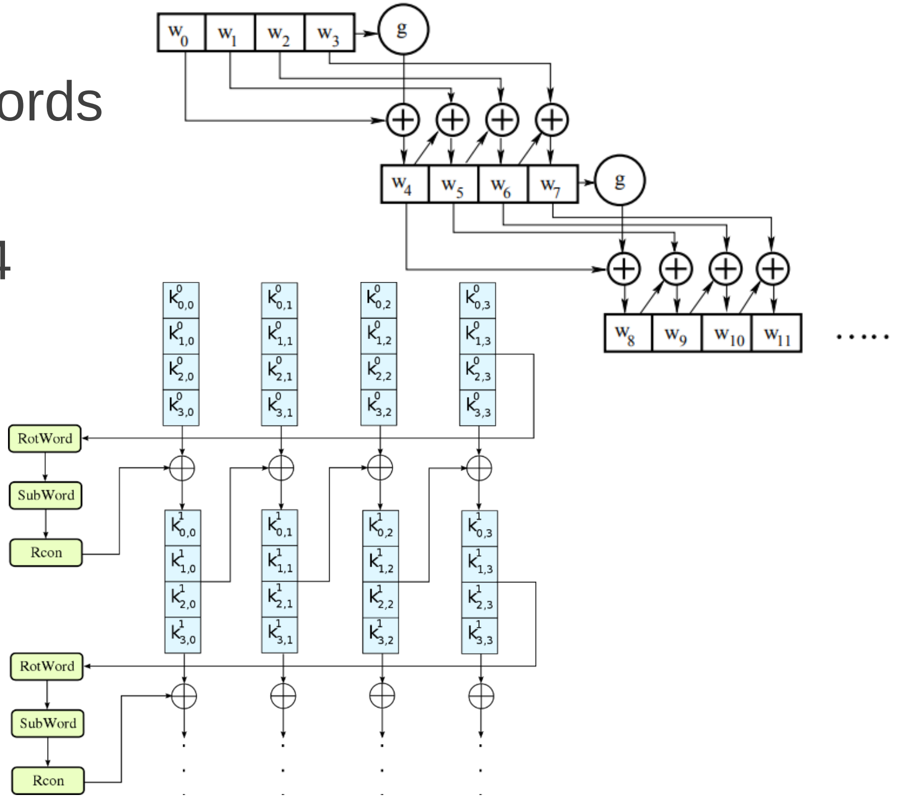


- XOR state with 128-bits of the round key
- again processed by column (though effectively a series of byte operations)
- inverse for decryption identical
  - since XOR own inverse, with reversed keys
- designed to be as simple as possible
  - requires other stages for complexity / security


$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,0}$ $s_{1,1}$ $s_{1,2}$ $s_{1,3}$
$s_{2,0}$ $s_{2,1}$ $s_{2,2}$ $s_{2,3}$
$s_{3,0}$ $s_{3,1}$ $s_{3,2}$ $s_{3,3}$
$\oplus$
w_i w_{i+1} w_{i+2} w_{i+3}
=
$s'_{0,0}$ $s'_{0,1}$ $s'_{0,2}$ $s'_{0,3}$
$s'_{1,0}$ $s'_{1,1}$ $s'_{1,2}$ $s'_{1,3}$
$s'_{2,0}$ $s'_{2,1}$ $s'_{2,2}$ $s'_{2,3}$
$s'_{3,0}$ $s'_{3,1}$ $s'_{3,2}$ $s'_{3,3}$

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 is XORed with 4th that underwent rotate + S-box + XOR round constant on previous

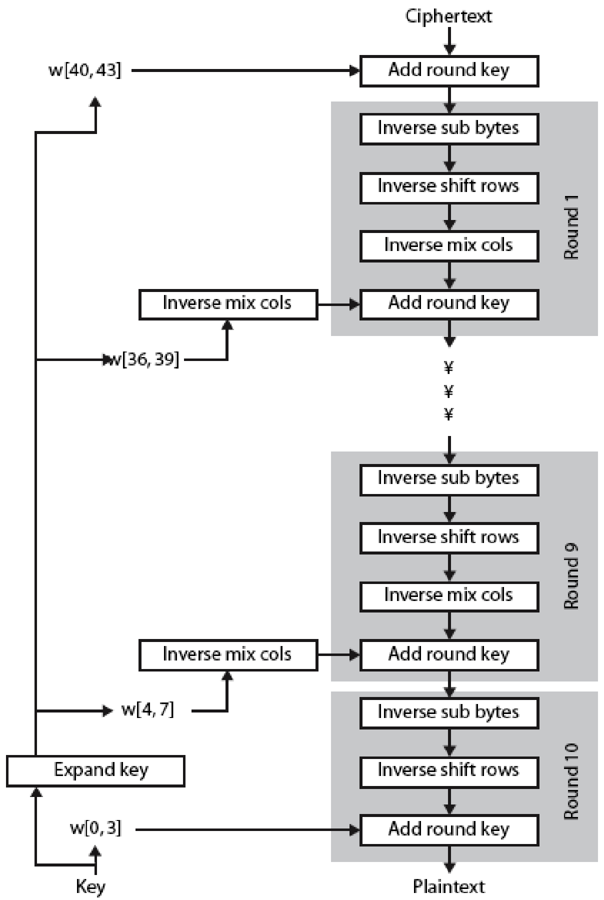

w_0
w_1
w_2
w_3
g
w_4
w_5
w_6
w_7
g
w_8
w_9
w_{10}
w_{11}
.....
$K^0_{0,0}$ $K^0_{1,0}$ $K^0_{2,0}$ $K^0_{3,0}$
$K^0_{0,1}$ $K^0_{1,1}$ $K^0_{2,1}$ $K^0_{3,1}$
$K^0_{0,2}$ $K^0_{1,2}$ $K^0_{2,2}$ $K^0_{3,2}$
$K^0_{0,3}$ $K^0_{1,3}$ $K^0_{2,3}$ $K^0_{3,3}$
RotWord
SubWord
Rcon
$K^1_{0,0}$ $K^1_{1,0}$ $K^1_{2,0}$ $K^1_{3,0}$
$K^1_{0,1}$ $K^1_{1,1}$ $K^1_{2,1}$ $K^1_{3,1}$
$K^1_{0,2}$ $K^1_{1,2}$ $K^1_{2,2}$ $K^1_{3,2}$
$K^1_{0,3}$ $K^1_{1,3}$ $K^1_{2,3}$ $K^1_{3,3}$
RotWord
SubWord
Rcon

Key Expansion Rationale

- designed to resist known attacks
- design criteria included
  - knowing part of 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

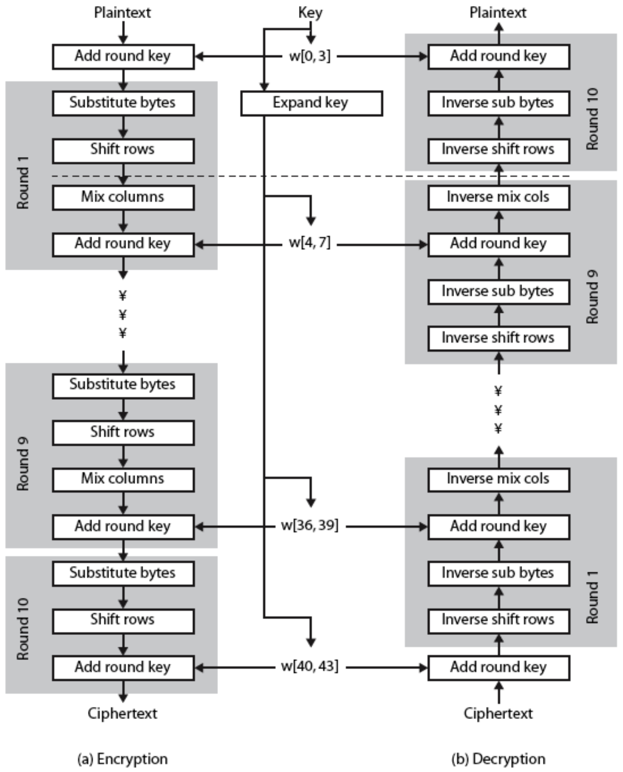
- AES decryption is not identical to encryption since steps done in reverse
- but can define an equivalent inverse cipher with steps as for encryption
  - but using inverses of each step
  - with a different key schedule
- works since result is unchanged when
  - swap byte substitution & shift rows
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

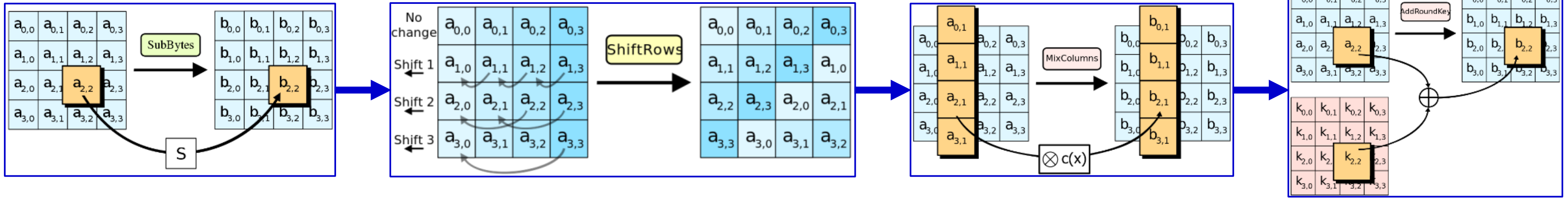


Recall: One AES Round

Remember

- Each round consists of 4 processes
- All n rounds are applied to one block of plaintext!
- For a nice detailed explanation of AES, refer to: https://engineering.purdue.edu/kak/compsec/NewLectures/Lecture8.pdf



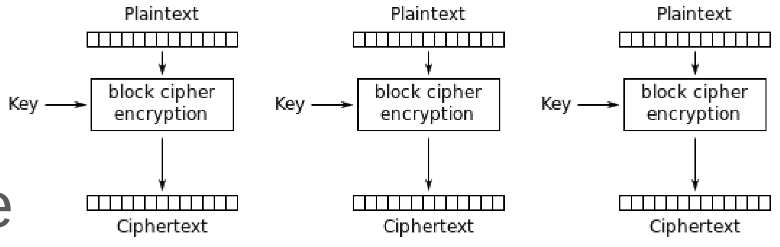


Modes of Operation

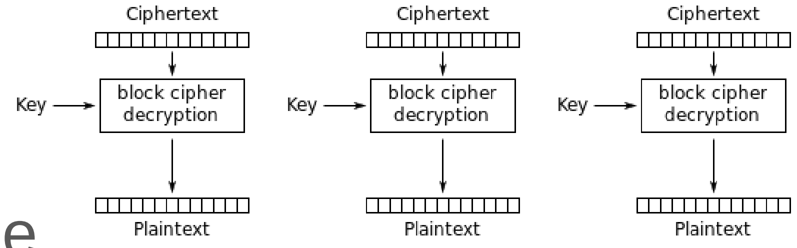
- Block ciphers encrypt fixed size blocks
  - eg. DES encrypts 64-bit blocks with 56-bit key
- Need some way to en/decrypt arbitrary amounts of data in practise
- **ANSI X3.106-1983 Modes of Use** (now FIPS 81) defines 4 possible modes
- Subsequently 5 defined for AES & DES
- Have **block** and **stream**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

Electronic Codebook Mode (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
  - $C_i = DES_{K1}(P_i)$
- Uses: secure transmission of single values



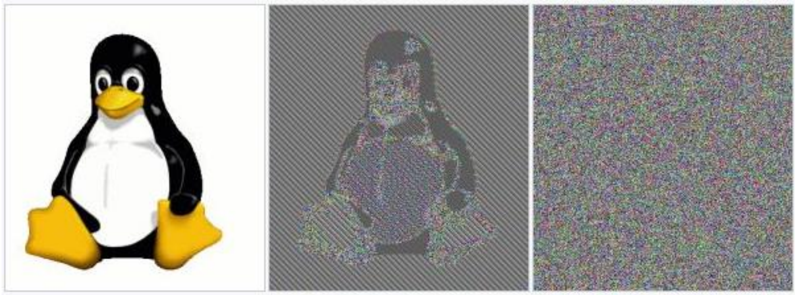
Electronic Codebook (ECB) mode encryption



Electronic Codebook (ECB) mode decryption

Limitations of ECB

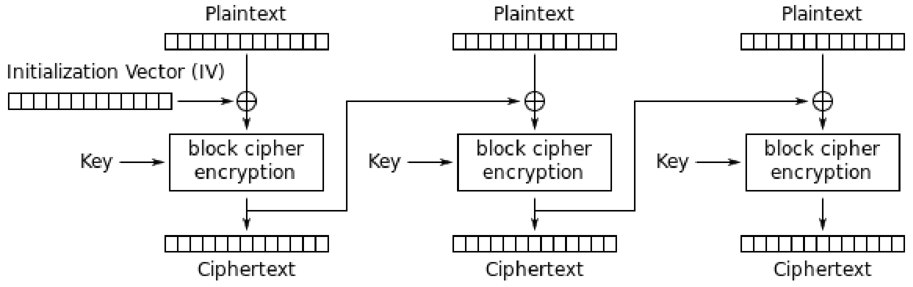
- Message repetitions may show in ciphertext
  - if aligned with message block
  - particularly with data such as graphics
  - or with messages that change very little, which become a code-book analysis problem
- Weakness is due to the encrypted message blocks being independent
- Main use is sending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



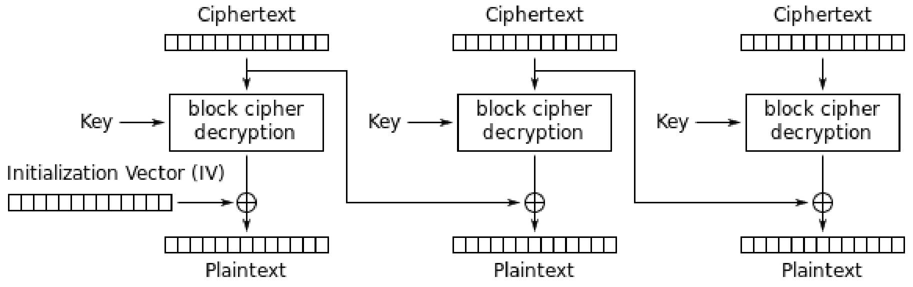
Original image | Encrypted using ECB mode | Modes other than ECB result in pseudo-randomness

Cipher Block Chaining Mode (CBC)

- Message is broken into blocks
- Linked together in encryption operation
- Each previous cipher block is chained with current plaintext block, hence name
- Use **Initial Vector (IV)** to start process
  - $C_i = DES_{K1}(P_i \text{ XOR } C_{i-1})$
  - $C_0 = IV$
- Uses: bulk data encryption, authentication



Cipher Block Chaining (CBC) mode encryption



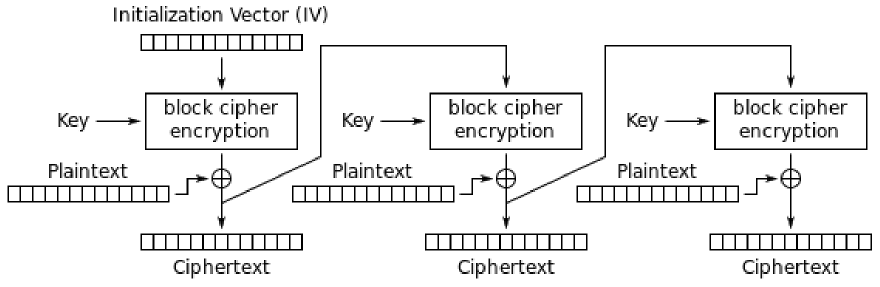
Cipher Block Chaining (CBC) mode decryption

Limitations of CBC

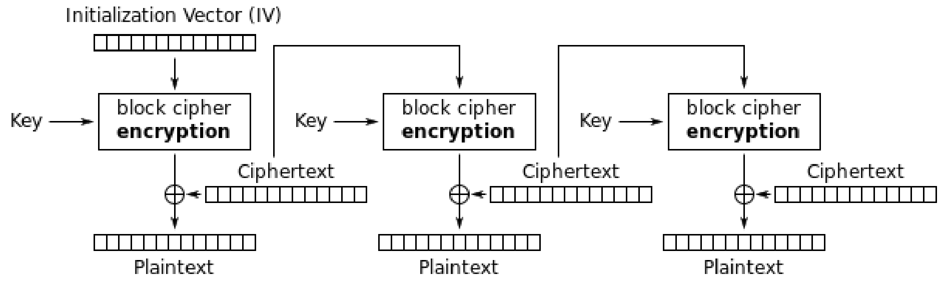
- A ciphertext block depends on **all** blocks before it
- Any change to a block affects all following ciphertext blocks
- Need **Initialization Vector** (IV)
  - which must be known to sender & receiver
  - if predictable, attacker can change bits of first block, and change IV to compensate
    - $C_1 = E(K, [IV \oplus P_1])$
    - $P_1 = IV \oplus D(K, C_1)$
    - $P_1[i] = IV[i] \oplus D(K, C_1)[i]$
    - $P_1[i]' = IV[i]' \oplus D(K, C_1)[i]$
  - hence IV must be an unpredictable value
  - can be sent encrypted in ECB mode before rest of message

Cipher Feedback Mode (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 bits (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)
  - $C_i = P_i \text{ XOR } DES_{K1}(C_{i-1})$
  - $C_0 = IV$
- Uses: stream data encryption, authentication



Cipher Feedback (CFB) mode encryption

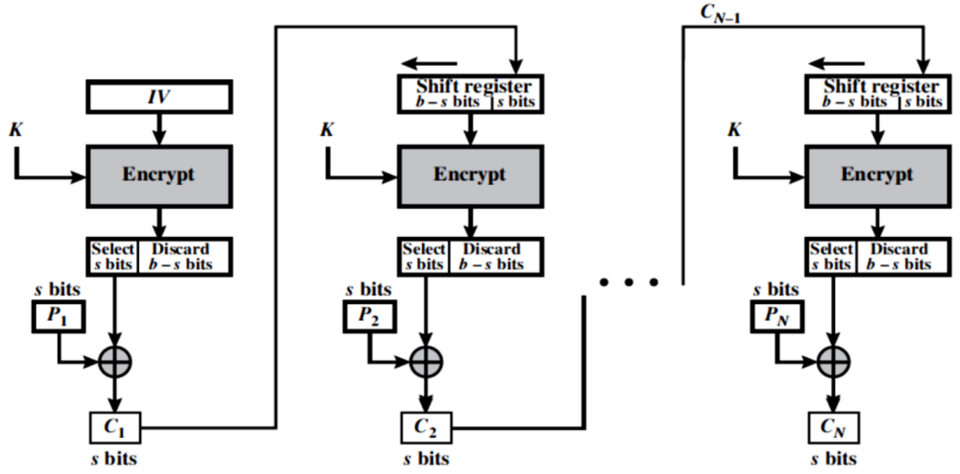


Cipher Feedback (CFB) mode decryption

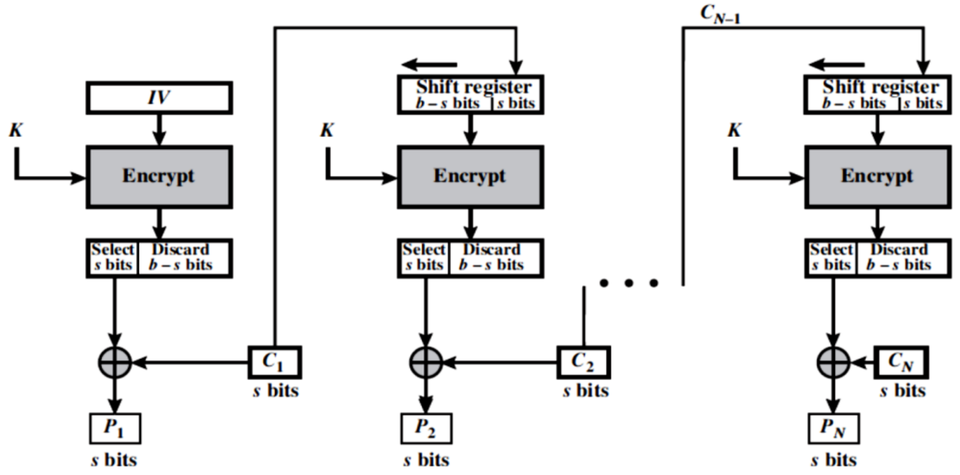
Cipher Feedback Mode (CFB)

The use of shift registers to enable self-synchronization

- If x bits are lost from the ciphertext, the cipher will output incorrect plaintext until the shift register once again equals a state it held while encrypting, at which point the cipher has resynchronized



(a) Encryption



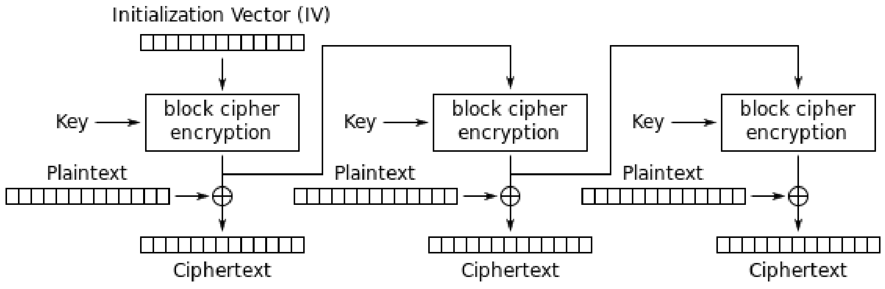
(b) Decryption

Limitations of CFB

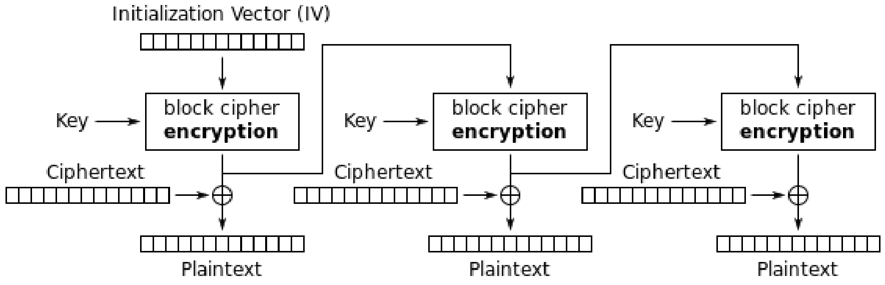
- Appropriate when data arrives in bits/bytes
- Most common stream mode
- Limitation is need to stall while doing block encryption after every n-bits
- Note that the block cipher is used in **encryption** mode at **both** ends
- Errors propagate for several blocks after the error

Output Feedback Mode (OFB)

- Message is treated as a stream of bits
- Output of cipher is added to message
- Output is then feed back (hence name)
- Feedback is independent of message
- Can be computed in advance
  - $C_i = P_i \text{ XOR } O_i$
  - $O_i = DES_{K1}(O_{i-1}) \quad i>1$
  - $O_1 = DES(Nonce)$
- Uses: stream encryption on noisy channels



Output Feedback (OFB) mode encryption



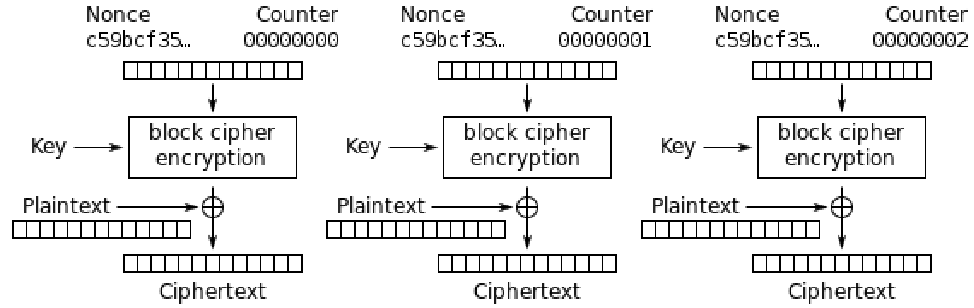
Output Feedback (OFB) mode decryption

Limitations of OFB

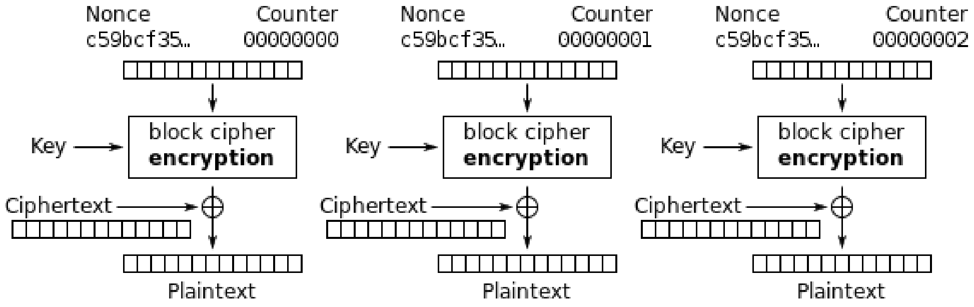
- Bit errors do not propagate
- More vulnerable to message stream modification
- A variation of a Vernam cipher
  - hence must **never** reuse the same sequence (key+IV)
- Sender & receiver must remain in sync
- Originally specified with m-bit feedback
- Subsequent research has shown that only **full block feedback** (i.e. CFB-64 or CFB-128) should ever be used

Counter Mode (CTR)

- Relatively "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)
  - $C_i = P_i \text{ XOR } O_i$
  - $O_i = DES_{K1}(i)$
- Uses: high-speed network encryptions



Counter (CTR) mode encryption



Counter (CTR) mode decryption

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)

Mode	Description	Typical Application
Electronic Codebook (ECB)	Each block of 64 plaintext bits is encoded independently using the same key.	• Secure transmission of single values (e.g., an encryption key)
Cipher Block Chaining (CBC)	The input to the encryption algorithm is the XOR of the next 64 bits of plaintext and the preceding 64 bits of ciphertext.	• General-purpose block-oriented transmission • Authentication
Cipher Feedback (CFB)	Input is processed *s* bits at a time. Preceding ciphertext is used as input to the encryption algorithm to produce pseudorandom output, which is XORed with plaintext to produce next unit of ciphertext.	• General-purpose stream-oriented transmission • Authentication
Output Feedback (OFB)	Similar to CFB, except that the input to the encryption algorithm is the preceding encryption output, and full blocks are used.	• Stream-oriented transmission over noisy channel (e.g., satellite communication)
Counter (CTR)	Each block of plaintext is XORed with an encrypted counter. The counter is incremented for each subsequent block.	• General-purpose block-oriented transmission • Useful for high-speed requirements