Information Security
CS 3002

Dr. Haroon Mahmood

Assistant Professor

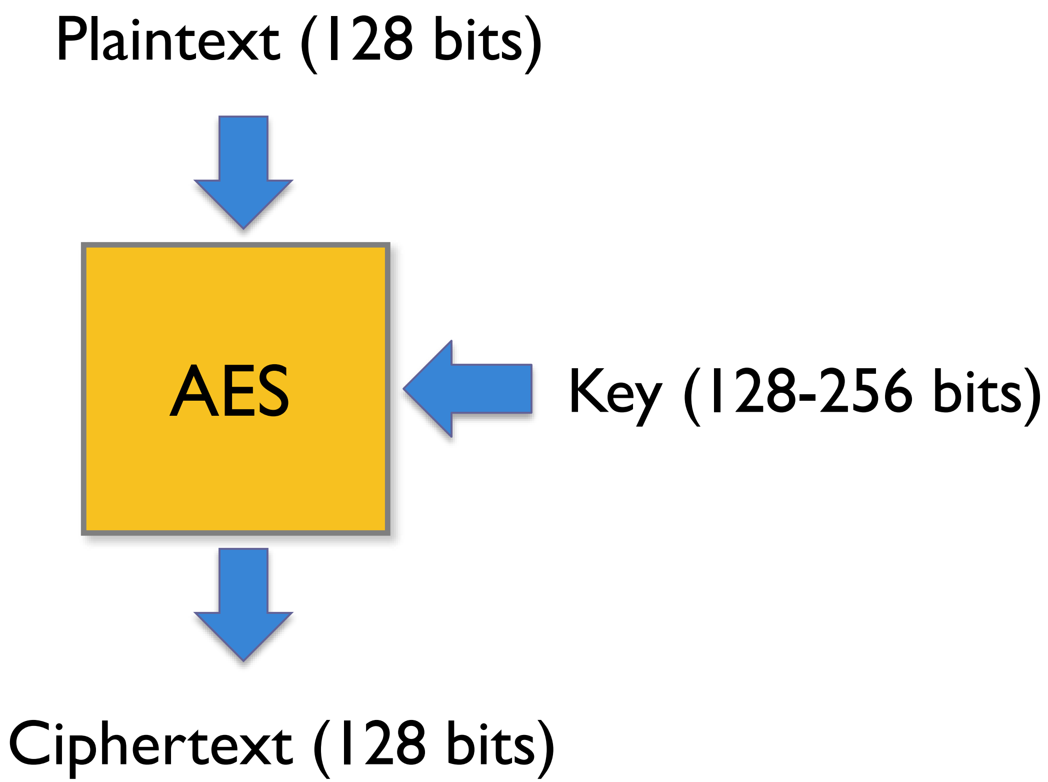
NUCES Lahore

The AES Cipher - Rijndael

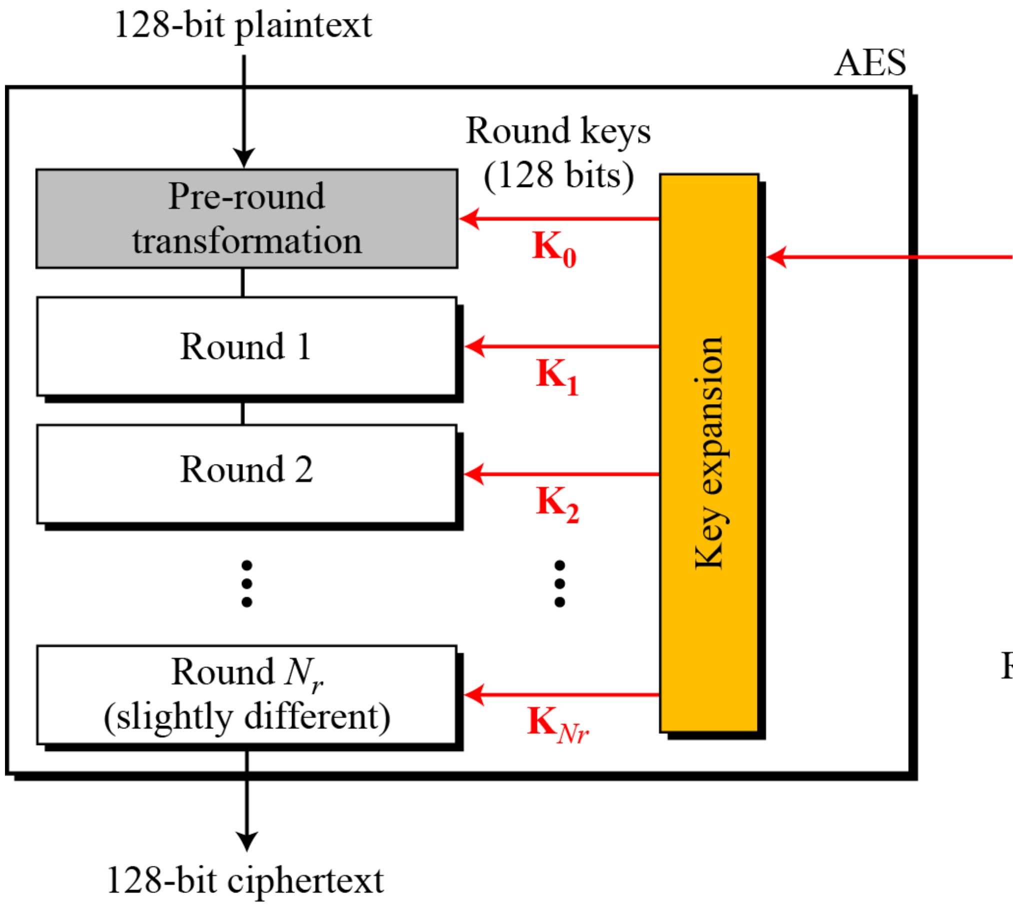
- An iterative rather than Feistel cipher
  - processes data as block of 4 columns of 4 bytes (128 bits)
  - operates on entire data block in every round

- Rijndael design has
  - 128/192/256 bit keys, 128 bits data
  - resistance against known attacks
  - speed and code compactness on many CPUs

AES Conceptual Scheme



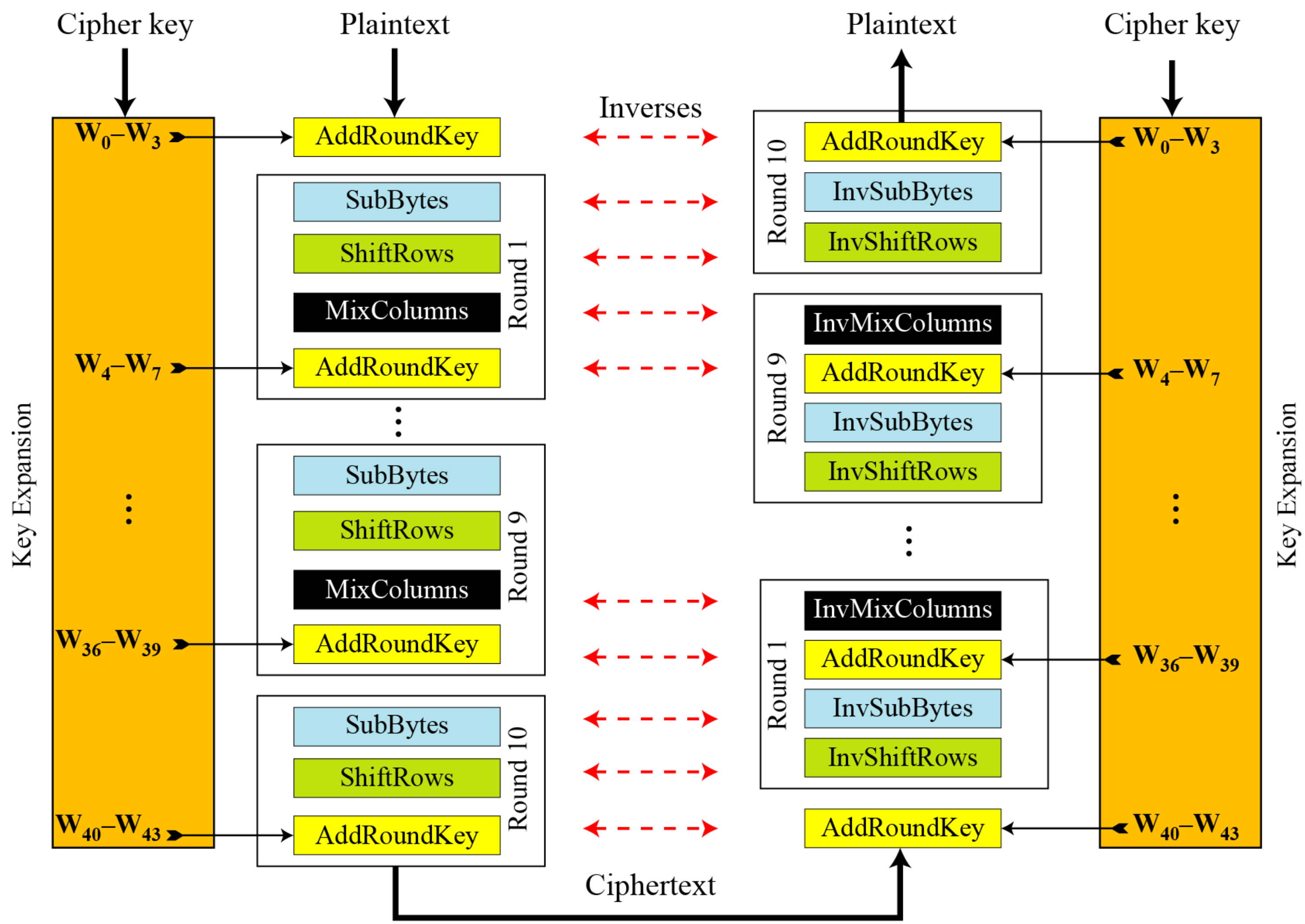
Multiple rounds



Nr	Key size
10	128
12	192
14	256

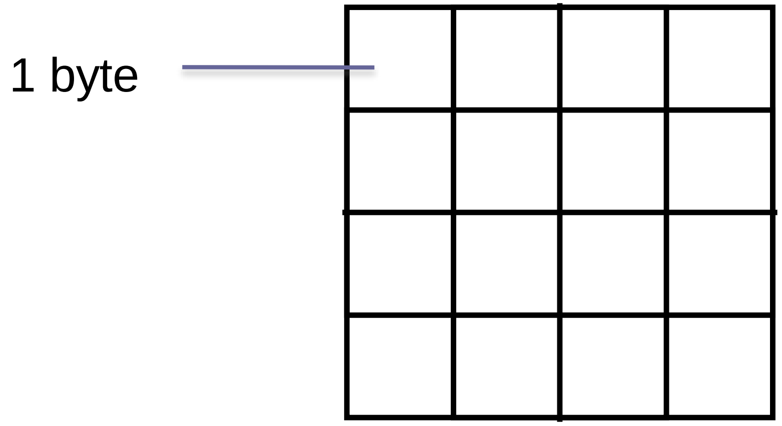
Relationship between number of rounds and cipher key size

Overall Structure



Structure of 128-bit block

- Data block viewed as table of bytes
- Represented as 4 by 4 matrix of bytes.
- Key is expanded to array of 32 bits words



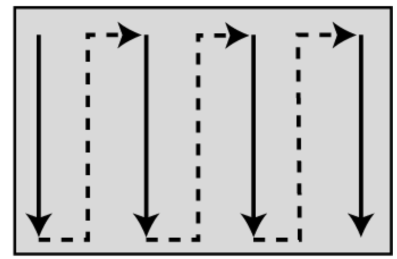
Data block represented as 'State'

b_0	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9	b_{10}	b_{11}	b_{12}	b_{13}	b_{14}	b_{15}

Block

(bytes)

$$\text{State}\quad \begin{bmatrix} s_{0,0}=b_0 & s_{0,1}=b_4 & s_{0,2}=b_8 & s_{0,3}=b_{12} \\ s_{1,0}=b_1 & s_{1,1}=b_5 & s_{1,2}=b_9 & s_{1,3}=b_{13} \\ s_{2,0}=b_2 & s_{2,1}=b_6 & s_{2,2}=b_{10} & s_{2,3}=b_{14} \\ s_{3,0}=b_3 & s_{3,1}=b_7 & s_{3,2}=b_{11} & s_{3,3}=b_{15} \end{bmatrix}$$

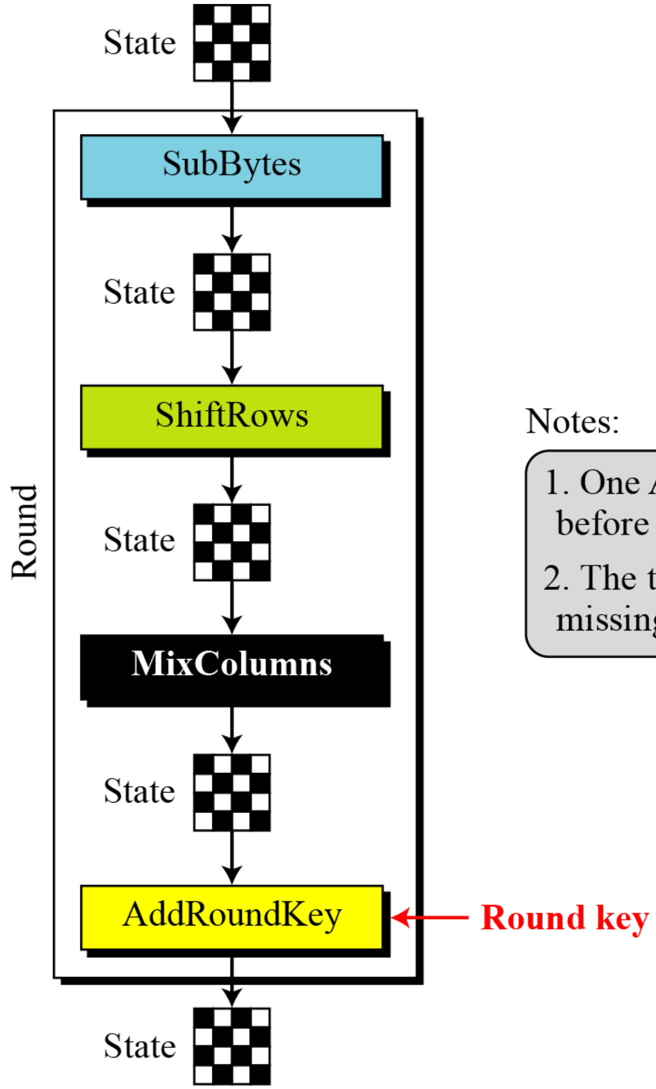


Example - changing plaintext to State

Text	A	E	S	U	S	E	S	A	M	A	T	R	I	X	Z	Z
Hexadecimal	00	04	12	14	12	04	12	00	0C	00	13	11	08	23	19	19

$$\begin{bmatrix} 00 & 12 & 0C & 08 \\ 04 & 04 & 00 & 23 \\ 12 & 12 & 13 & 19 \\ 14 & 00 & 11 & 19 \end{bmatrix} \quad \text{State}$$

Details of each round



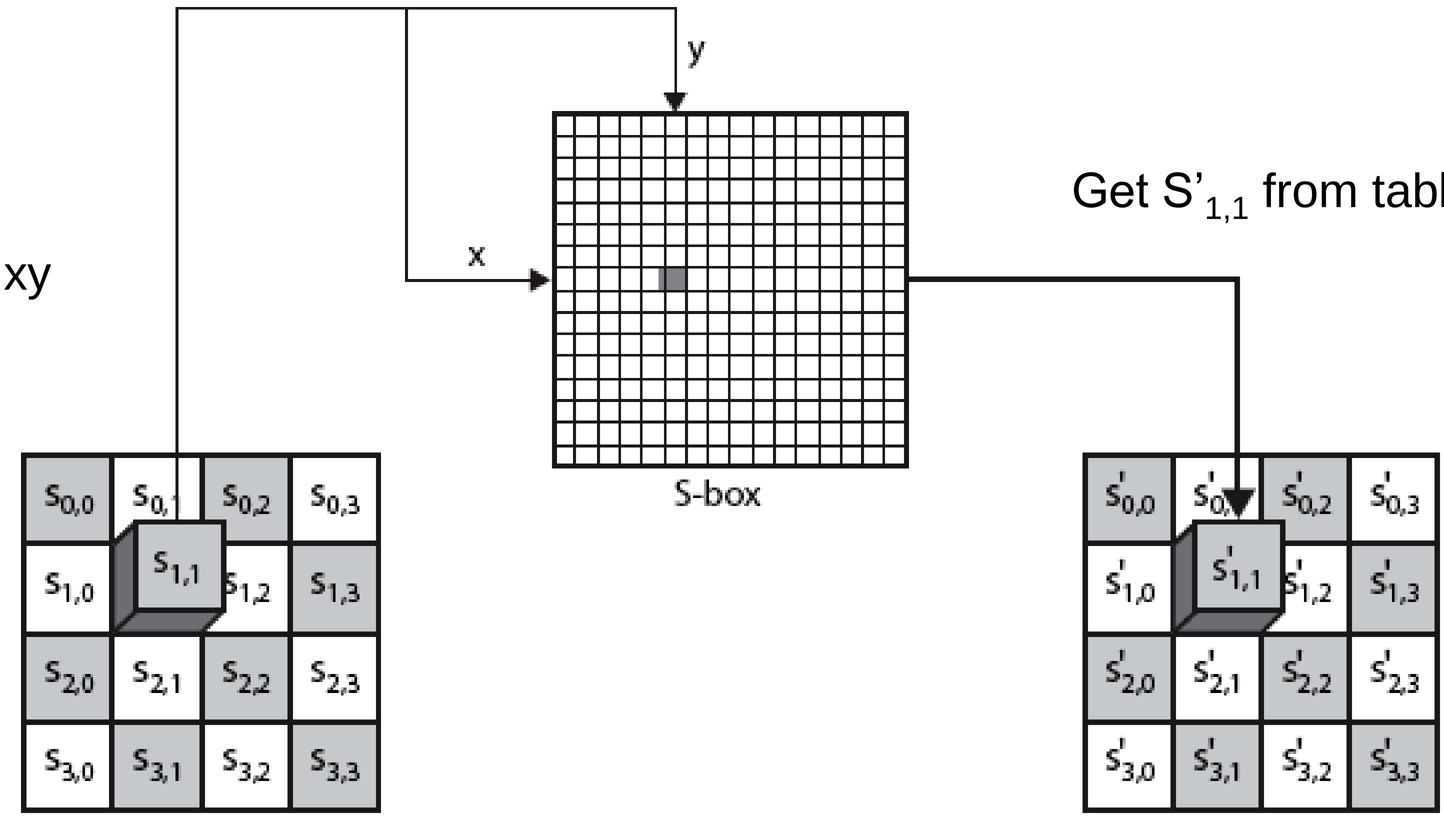
Notes:

1. One AddRoundKey is applied before the first round.
2. The third transformation is missing in the last round.

SubBytes Operation

The SubBytes operation involves 16 independent byte-to-byte transformations.

* Let $S_{1,1}$ = xy



Get $S'_{1,1}$ from table

*** Interpret a byte as two hexadecimal digits *xy***

SubBytes Table

	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

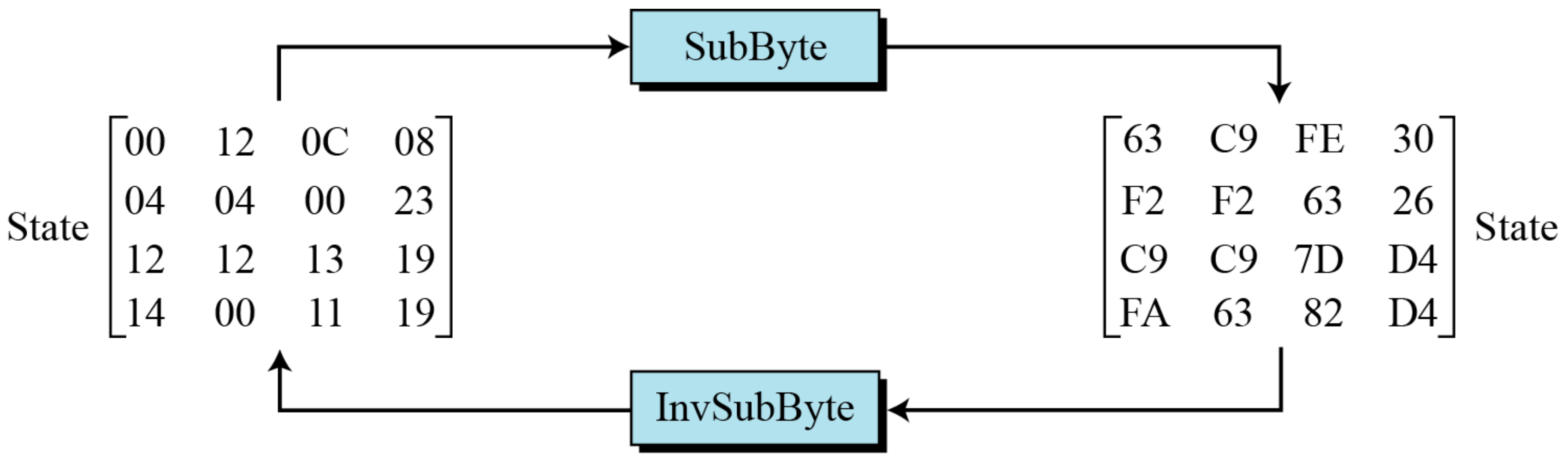
Implemented by table lookup

InvSubBytes Table

		y															
		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
x	**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

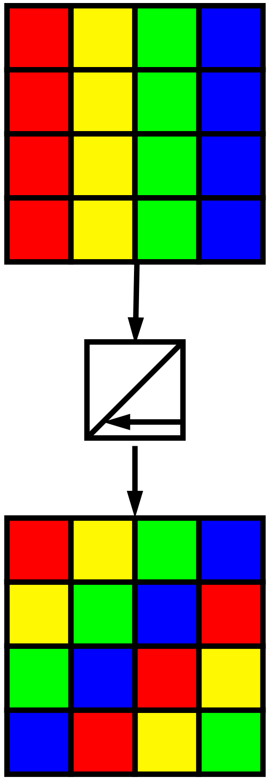
Sample SubByte Transformation

- **The SubBytes and InvSubBytes transformations are inverses of each other.**

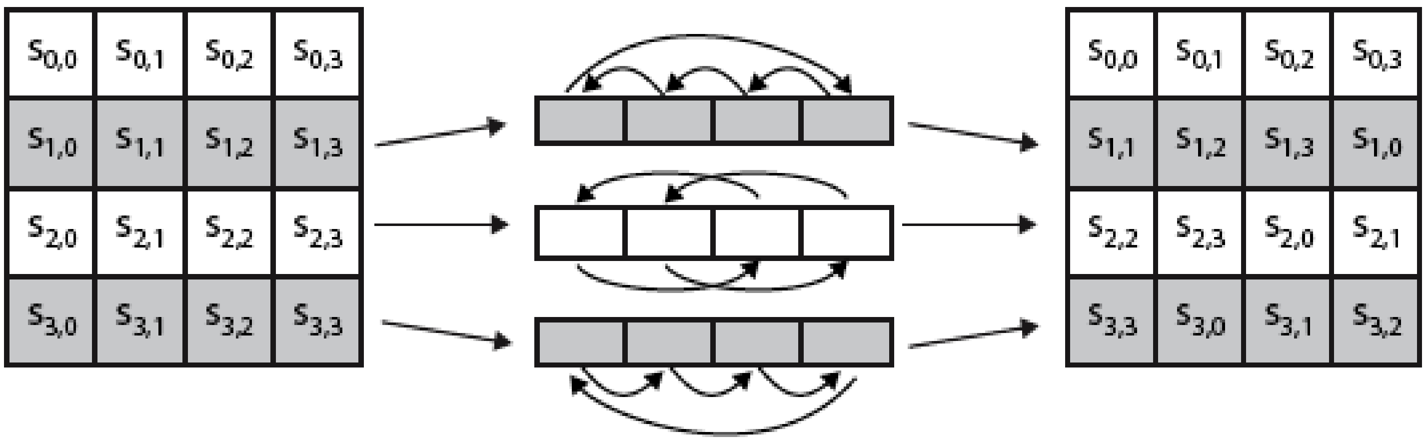


ShiftRows

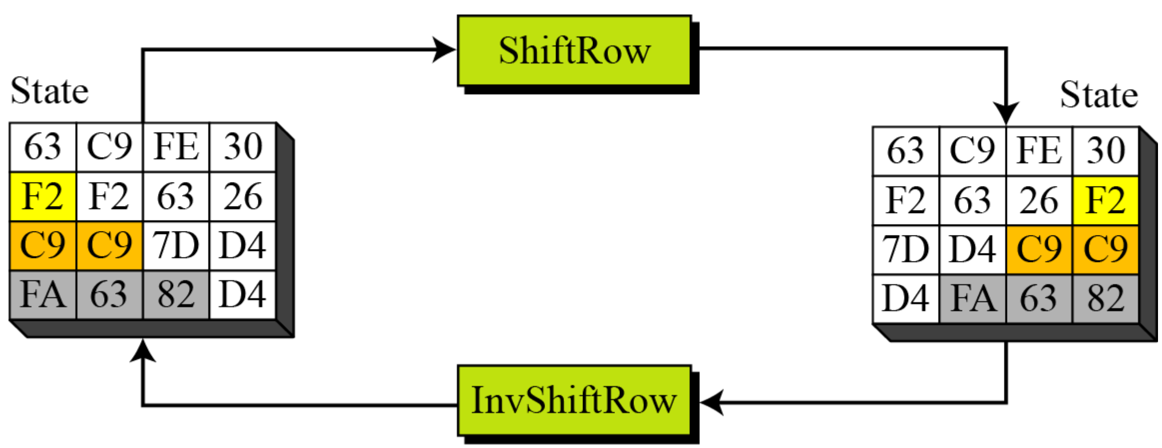
- Shifting, which permutes the bytes.
- A circular byte shift in each row
  - 1st row is unchanged
  - 2nd row does 1 byte circular shift to left
  - 3rd row does 2 byte circular shift to left
  - 4th row does 3 byte circular shift to left
- In the encryption, the transformation is called ShiftRows
- In the decryption, the transformation is called InvShiftRows and the shifting is to the right



ShiftRows Scheme



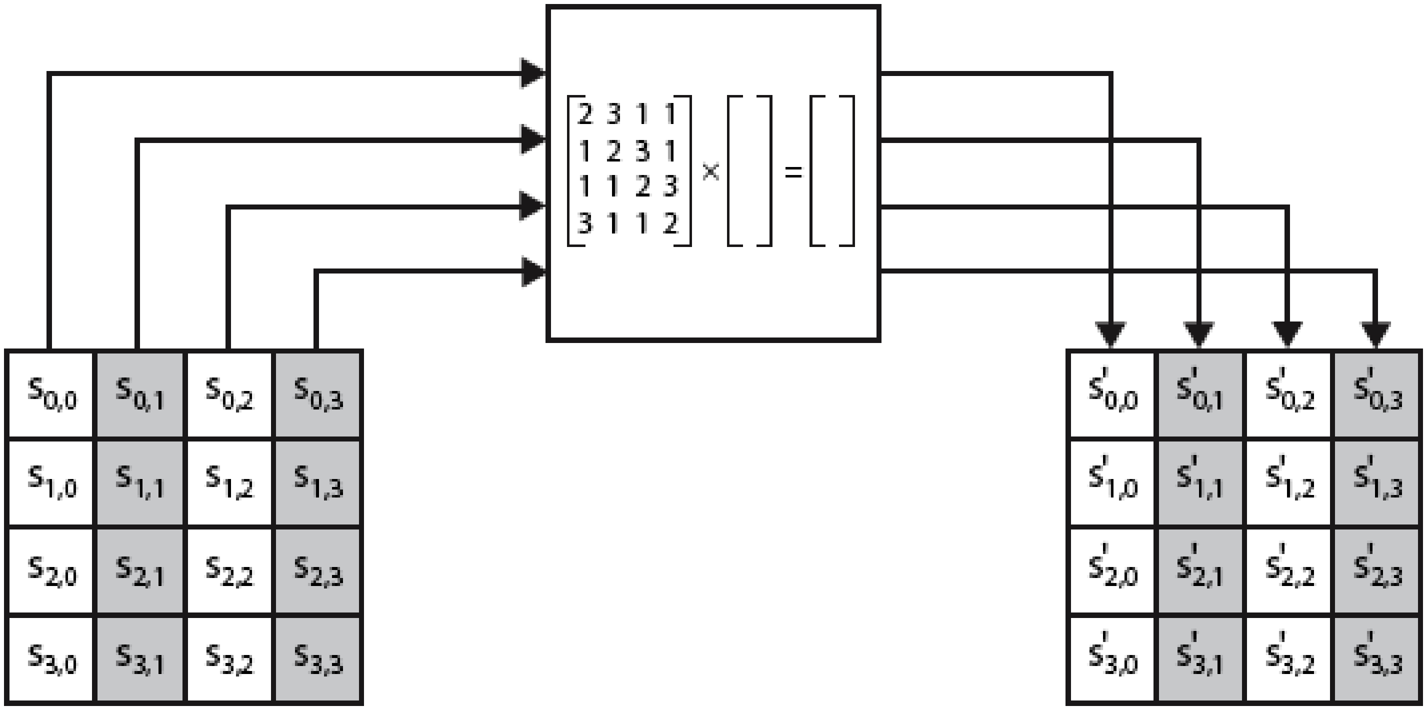
Example



MixColumns

- ShiftRows and MixColumns provide diffusion to the cipher
- In MixColumns, 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 finite field GF(2^8) using prime polynomial $x^8+x^4+x^3+x+1$

MixColumns Scheme



The MixColumns transformation operates at the column level; it transforms each column of the state to a new column.

MixColumn and InvMixColumn

During decryption, inverse mixing matrix is used

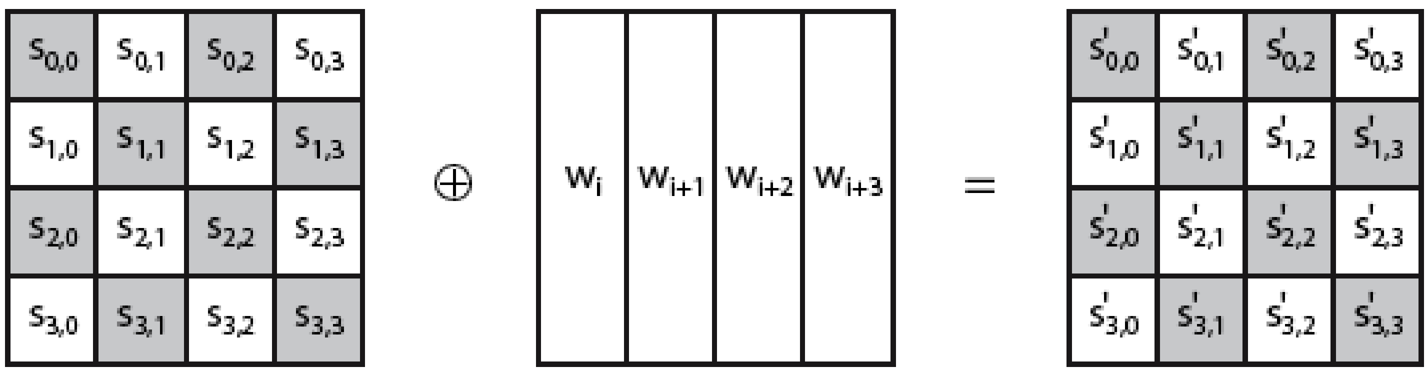
$$\underset{C}{\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix}} \xleftrightarrow{\text{Inverse}} \underset{C^{-1}}{\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix}}$$

AddRoundKey

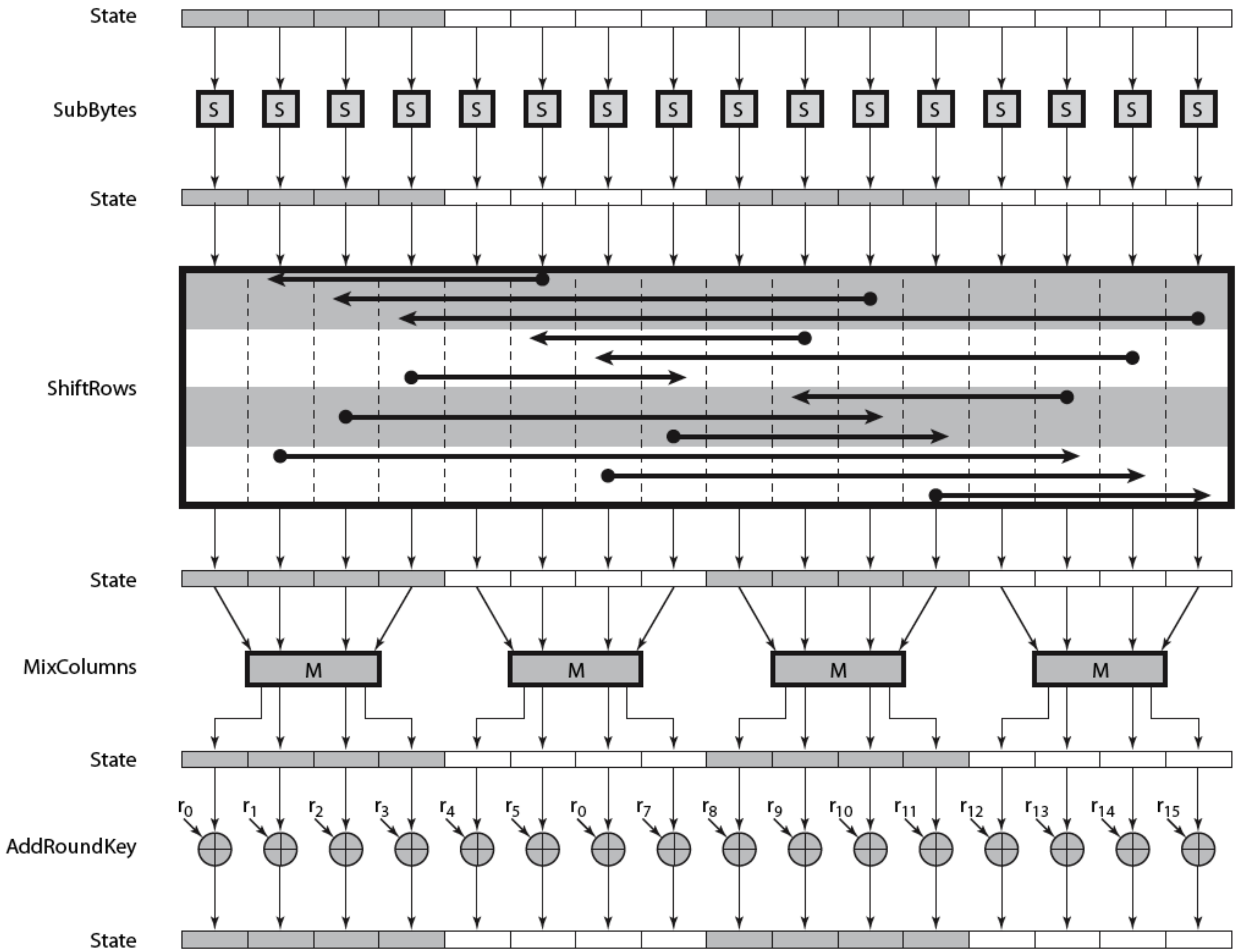
XOR state with 128-bits of the round key

- AddRoundKey proceeds one column at a time.
  - adds a round key word with each state column matrix
  - the operation is matrix addition
- Inverse for decryption is identical
  - since XOR is its own inverse, with same keys

AddRoundKey Scheme



AES Round



State
SubBytes
S
State
ShiftRows
State
MixColumns
M
State
AddRoundKey
r0
r1
r2
r3
r4
r5
r0
r7
r8
r9
r10
r11
r12
r13
r14
r15
State

AES Key Scheduling (generating round keys)

- **takes 128-bits (16-bytes) key and expands into array of 44 words (32-bit each)**

Round	*Words*			
Pre-round	$\mathbf{w}_0$	$\mathbf{w}_1$	$\mathbf{w}_2$	$\mathbf{w}_3$
1	$\mathbf{w}_4$	$\mathbf{w}_5$	$\mathbf{w}_6$	$\mathbf{w}_7$
2	$\mathbf{w}_8$	$\mathbf{w}_9$	$\mathbf{w}_{10}$	$\mathbf{w}_{11}$
. . .	. . .			
N_r	$\mathbf{w}_{4N_r}$	$\mathbf{w}_{4N_r+1}$	$\mathbf{w}_{4N_r+2}$	$\mathbf{w}_{4N_r+3}$

AES Key Scheduling (generating round keys)

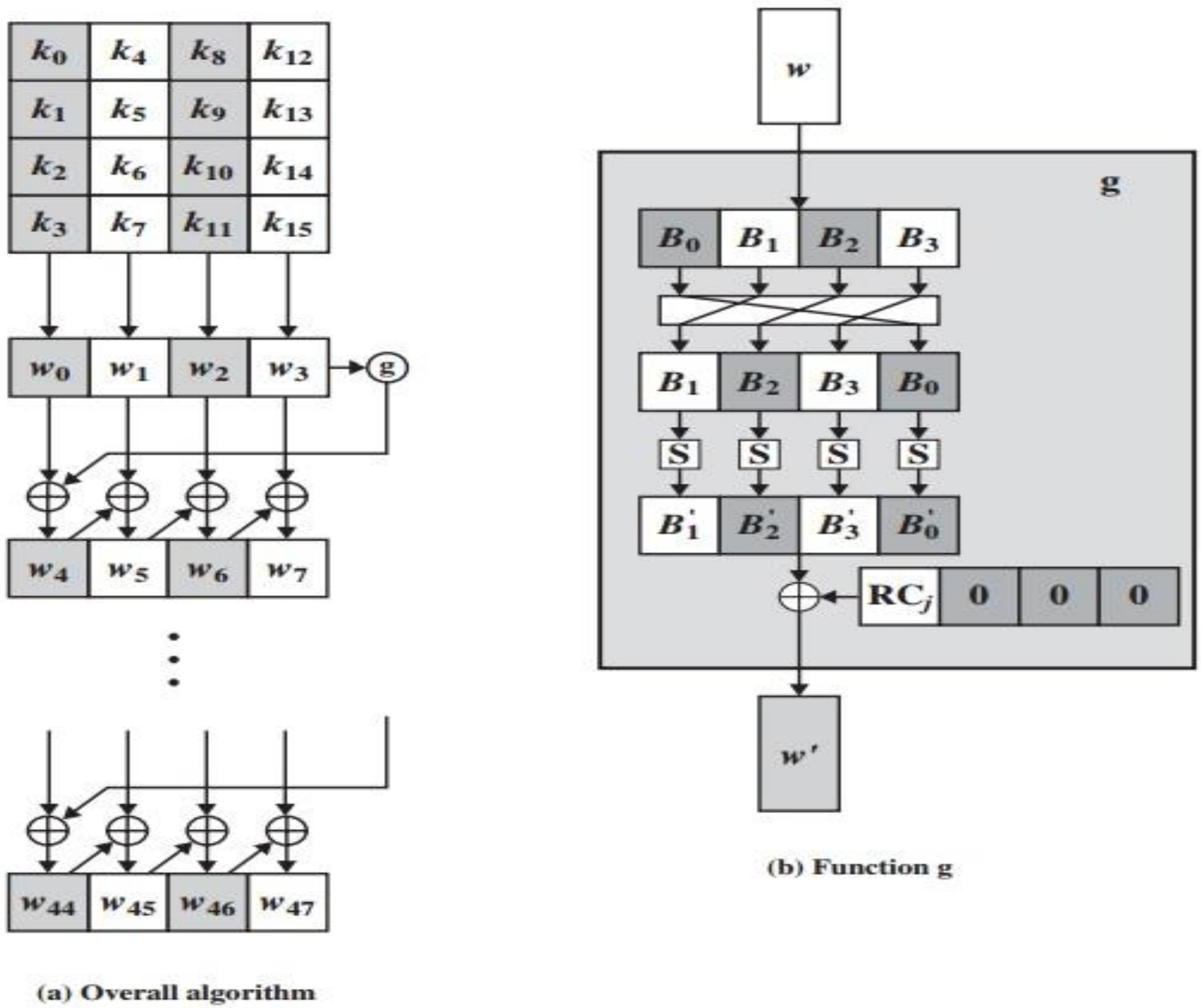


Figure 5.9 AES Key Expansion

AES Security

- AES was designed after DES.
  - Most of the known attacks on DES were already tested on AES.
- Brute-Force Attack
  - AES is definitely more secure than DES due to the larger-size key.
- Statistical Attacks
  - Numerous tests have failed to do statistical analysis of the ciphertext
- Differential and Linear Attacks
  - There are no differential and linear attacks on AES as yet.

Implementation Aspects

- The algorithms used in AES are so simple that they can be easily implemented using cheap processors and a minimum amount of memory.

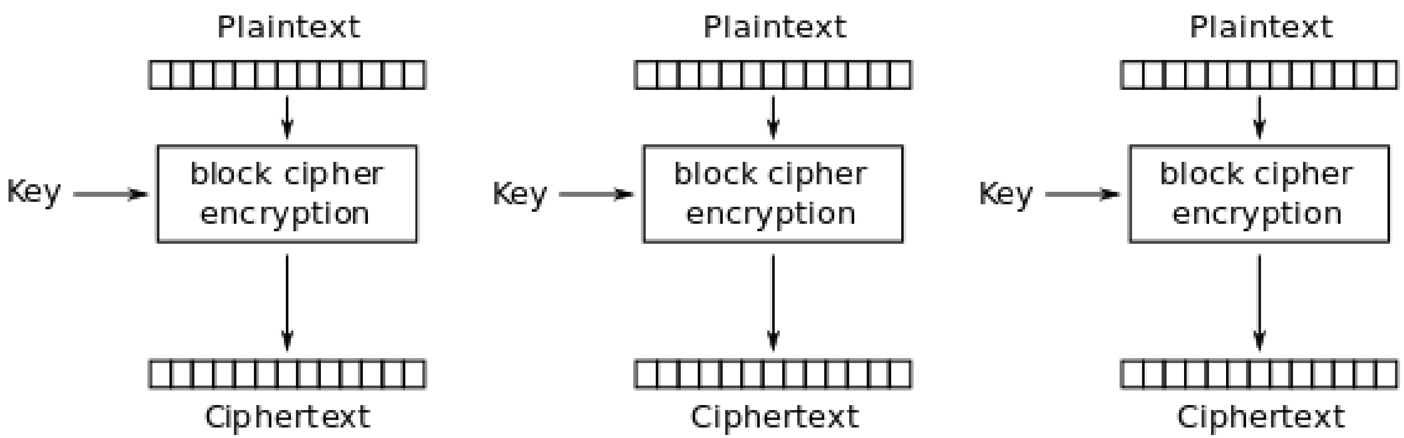
- Very efficient

- Implementation was a key factor in its selection as the AES cipher

- Several modern CPU architectures include AES instructions

Block Cipher modes of operation

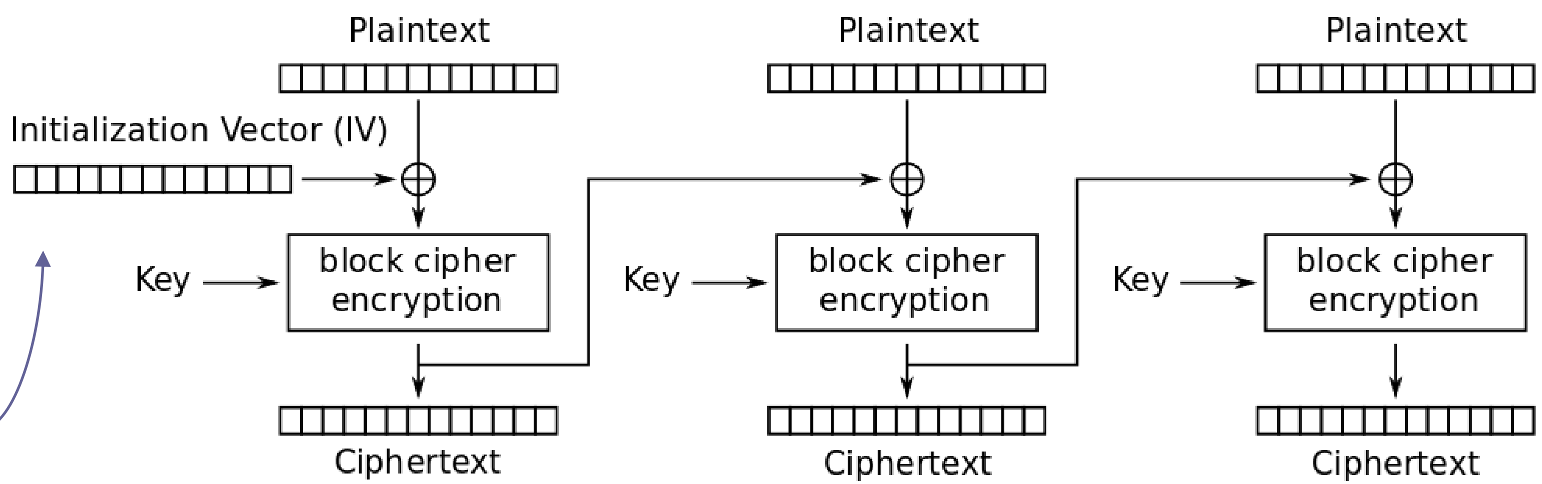
- Block ciphers operate only on small fixed size chunks like 64 bits (DES), 128 bits (AES) etc.
- To encrypt large data, one (lazy) option is to simply divide the whole data in blocks and encrypt them separately. This is called Electronic Code Book (ECB) mode with same key.



Electronic Codebook (ECB) mode encryption

Block Cipher modes of operation

- **More secure modes are available, such as Cipher Block Chaining**
- **Each ciphertext block depends on all plaintext blocks processed up to that point**

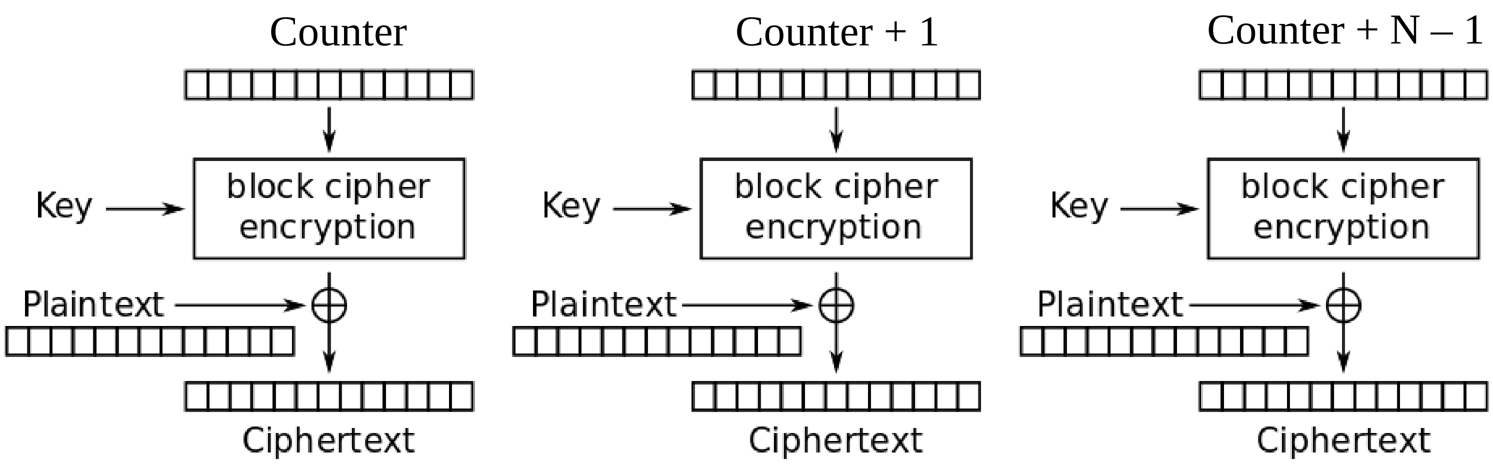


Cipher Block Chaining (CBC) mode encryption

Any fixed (non-secret) value to start with

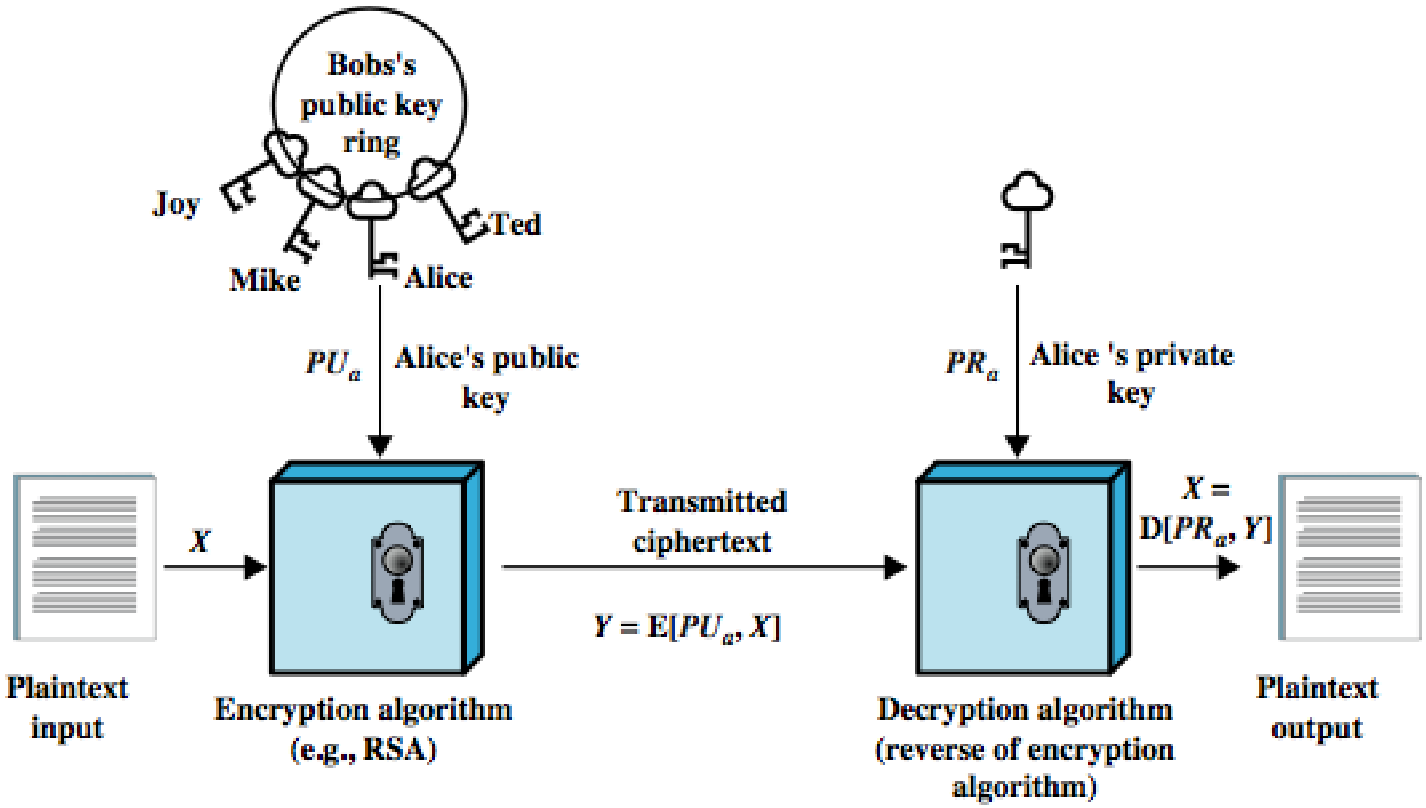
Block Cipher modes of operation

- **Another one is Counter mode**
- **Start with any pre-defined counter value and then keep incrementing.**
- **Can encrypt blocks in parallel (unlike CBC mode)**



Counter (CTR) mode encryption

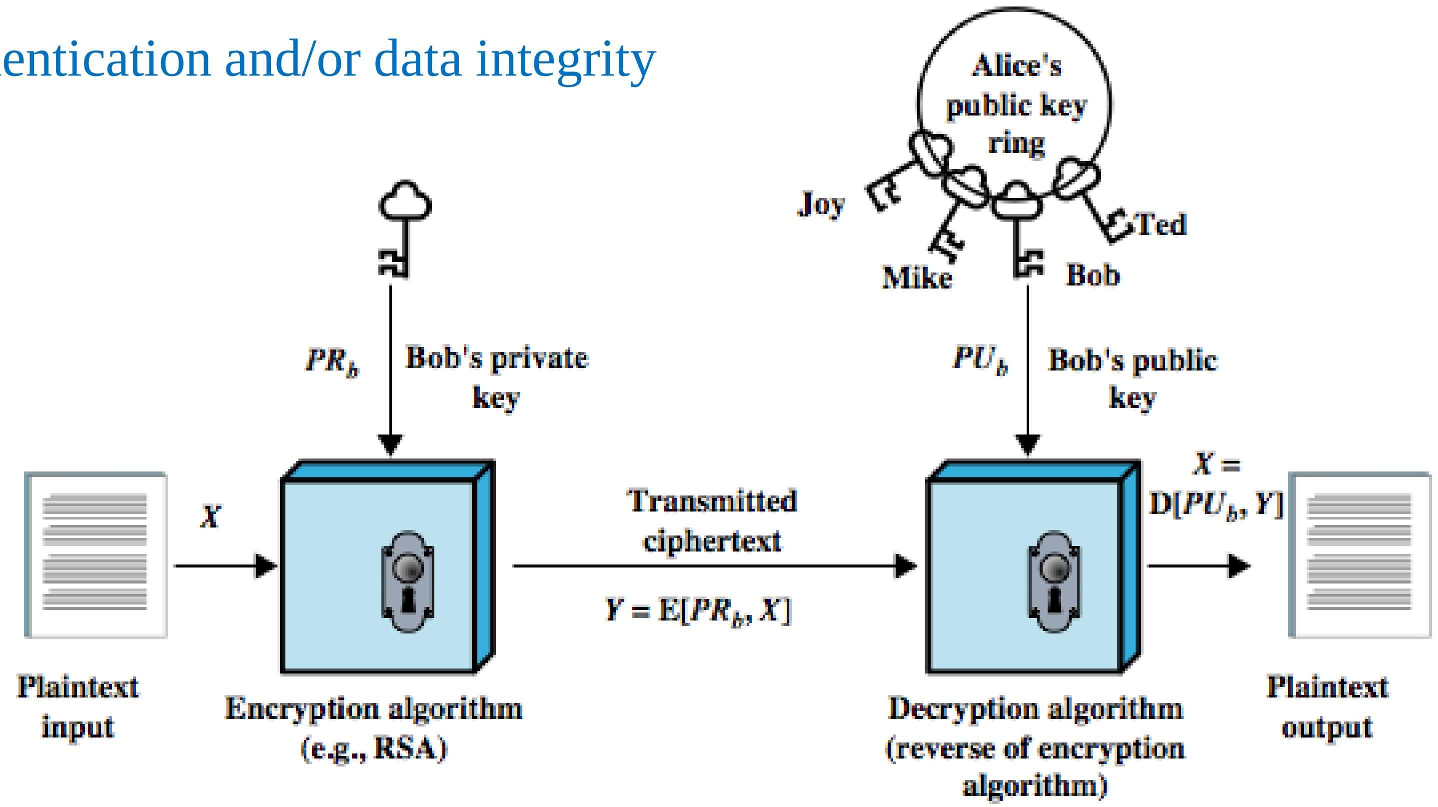
Public Key Encryption



(a) Confidentiality

Public Key Authentication

Authentication and/or data integrity



(b) Authentication

Public Key Requirements

1. computationally easy to create key pairs
2. computationally easy for sender knowing public key to encrypt messages
3. computationally easy for receiver knowing private key to decrypt ciphertext
4. computationally infeasible for opponent to determine private key from public key
5. computationally infeasible for opponent to otherwise recover original message
6. useful if either key can be used for each role

Public Key Algorithms

- RSA (Rivest, Shamir, Adleman)
  - developed in 1977
  - only widely accepted public-key encryption alg
  - given tech advances need 1024+ bit keys
- Diffie-Hellman key exchange algorithm
  - only allows exchange of a secret key
- Digital Signature Standard (DSS)
  - provides only a digital signature function with SHA-1
- Elliptic curve cryptography (ECC)
  - new, security like RSA, but with much smaller keys

Diffie-Hellman

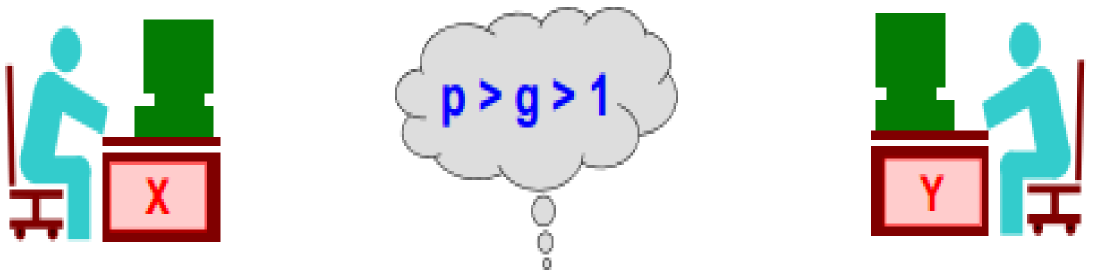
- First public key algorithm invented
- Published in 1976
- Specific method for securely exchanging cryptographic keys over a public channel
- Concept given by Ralph Merkle
- Named after Whitfield Diffie and Martin Hellman

- Public key exchange algorithm, neither encryption nor signature

Diffie-Hellman

- Uses modular arithmetic also called the clock arithmetic:
  - g mod p. where g is the generator and p is the prime modulus
  - $1 < g < p$
- g is a primitive root of p
- Consider two numbers g & p shared publically between A & B
- A computes $X = g^x \bmod p$ (x is the secret from Alice)
- B computes $Y = g^y \bmod p$ (y is the secret from Bob)
- A & B exchanged X & Y
- A computes $K_{AB} = Y^x \bmod p$ and B computes $K_{BA} = X^y \bmod p$
- $K_{AB} = K_{BA} = g^{xy} \bmod p$

Diffie-Hellman



$A = g^X \bmod p$ $\qquad$ $B = g^Y \bmod p$

$$\xrightarrow{\quad A \quad}$$

$$\xleftarrow{\quad B \quad}$$

$K_A = B^X \bmod p$ $\qquad$ $K_B = A^Y \bmod p$

$$\boxed{K_A = K_B = g^{XY} \bmod p = K_{AB}}$$

Diffie-Hellman: example

- E.g: $g = 3$, $p = 353$

- Alice computes secret => $x = 97$
  $g^x \bmod p = 3^{97} \bmod 353 = 40$
- Bob computes secret => $y = 233$
- $g^y \bmod p = 3^{233} \bmod 353 = 248$

- Alice gets 248 from Bob =>
- Alice computes => $248^{97} \bmod 353 = 160$
- Bob gets 40 from Alice =>
- Bob computes => $40^{233} \bmod 353 = 160$