Advance Encryption Standard:AES

- Block cipher concept
- non-linear function (Affine transform)
- S-box P-box SP network
- AES Subtsitute Shift-row Mix-column
- AES mode

Read

- Chapter 5-6, W.Stalling “Cryptography and Network security”
- Chapter 4, C.Paar “Understanding cryptography”
- บท 4 กฤดากร, วิทยาการรหัสลับ

Symmetric (secret) Key

- Alice and Bob share a secret key, K_{ab}
- Encryption – Plaintext message is encrypted and decrypted with K_{ab}
- The key is shared via a” secure Channel”
- Use for Bulk encryption
- Two types; Block ,Stream cipher

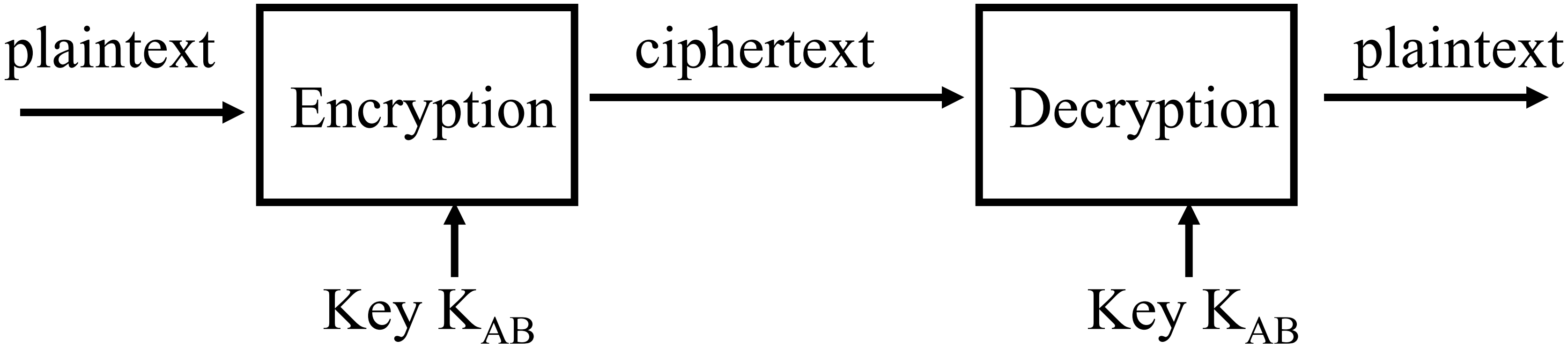
Kerckhoffs’ Principle

The system must be practically

It **must not be required to be secret**, and it must be able to fall into the hands of the enemy

Its key must be communicable.

Principles of Confusion and Diffusion



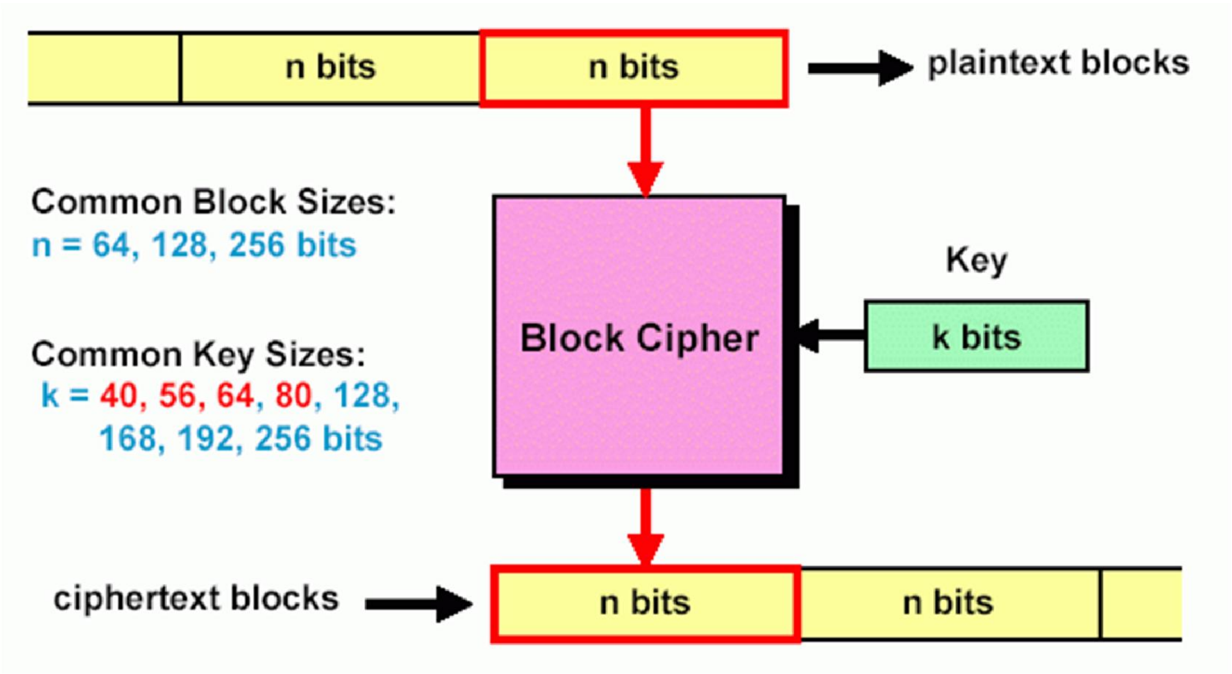
- Terms courtesy of Claude Shannon, father of Information Theory
- “Confusion” = Substitution
- “Diffusion” = Transposition or Permutation

Block cipher

- Block cipher is an encryption function that works on fixed size blocks
- Current block size is 128 bits
- Encrypting a 128-bit plaintext block produces a 128-bit ciphertext block
- The encryption key is also a series of bits, usually 128 or 256 bits
- A secure block cipher is one that keeps the plaintext secret

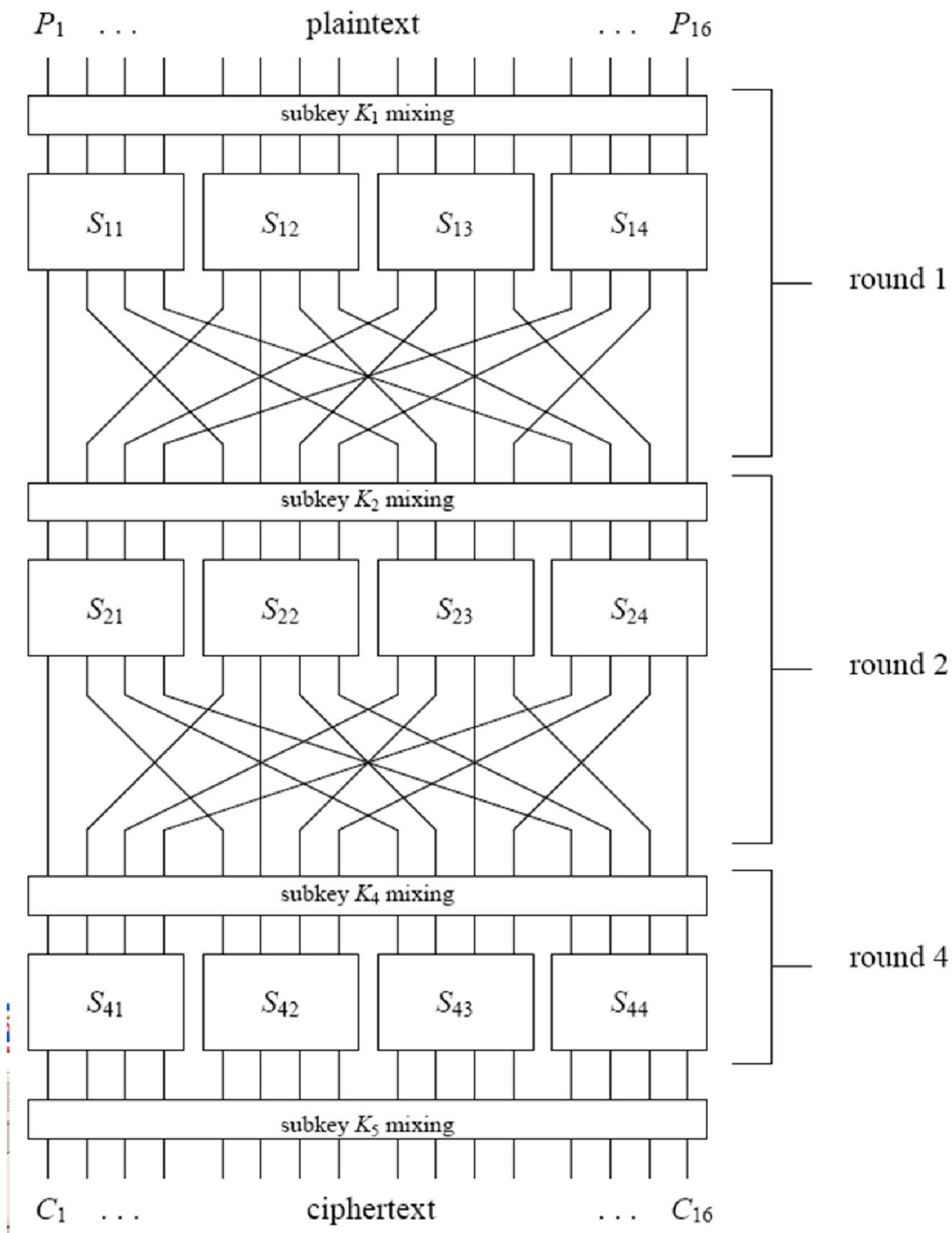
Block cipher

Divide input bit stream into n-bit sections, encrypt only that section, no dependency/history between sections



In a good block cipher, each output bit is a function of all n input bits and all k key bits

Substitute Permute Network



A Simple SPN

For example: a 16-bit SPN with 16-bit key that uses the following:

Substitution 4 bit

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

Permutation 3bit

Input permute	123	132	213	231	312	321
inverse	123	132	213	312	231	321

Can we use substitutions that are Not bijections (1 to 1 function)?

Advanced Encryption Standard (Rijndahl)

- Replaces DES
- Selected by competition by NIST in 2001
- Reviewed by NSA and approved for classified data in 2003
- 128 bit block size
- 128, 192, or 256 bit key
- 10, 12, or 14 rounds of a substitution-permutation network

http://www.formaestudio.com/rijndaelinspector/

The Rijndael Animation

AES operation

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 row (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

Finite Fields

- AES uses the finite field $GF(2^8)$
  - $b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0$
    - $\{b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0\}$
- Byte notation for the element: $x^6 + x^5 + x + 1$
  - {01100011} – binary
  - {63} – hex
- Arithmetic operations
  - Addition
  - Multiplication

  Under $GF(2^8)$ with generator polynomial

  $x^8 + x^4 + x^3 + x + 1$

Efficient Finite field Multiply

Example: $\{57\} \bullet \{13\}$,(13=0001 0011)

$\underline{\{57\} \bullet \{02\}} = \mathrm{xtime}(\{57\}) = \{ae\}$

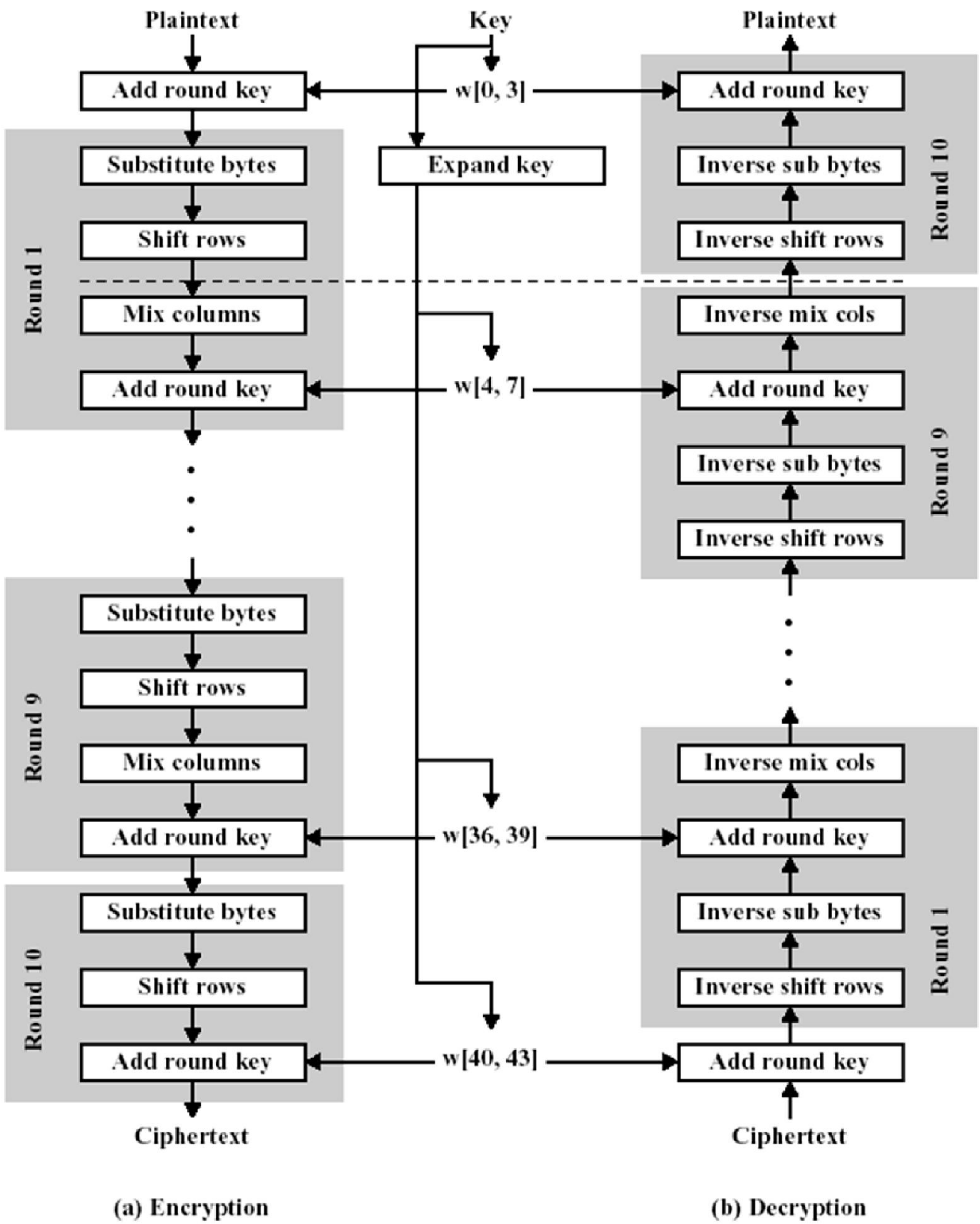
$\{57\} \bullet \{04\} = \mathrm{xtime}(\{ae\}) = \{47\}$

$\{57\} \bullet \{08\} = \mathrm{xtime}(\{47\}) = \{8e\}$

$\underline{\{57\} \bullet \{10\}} = \mathrm{xtime}(\{8e\}) = \{07\}$

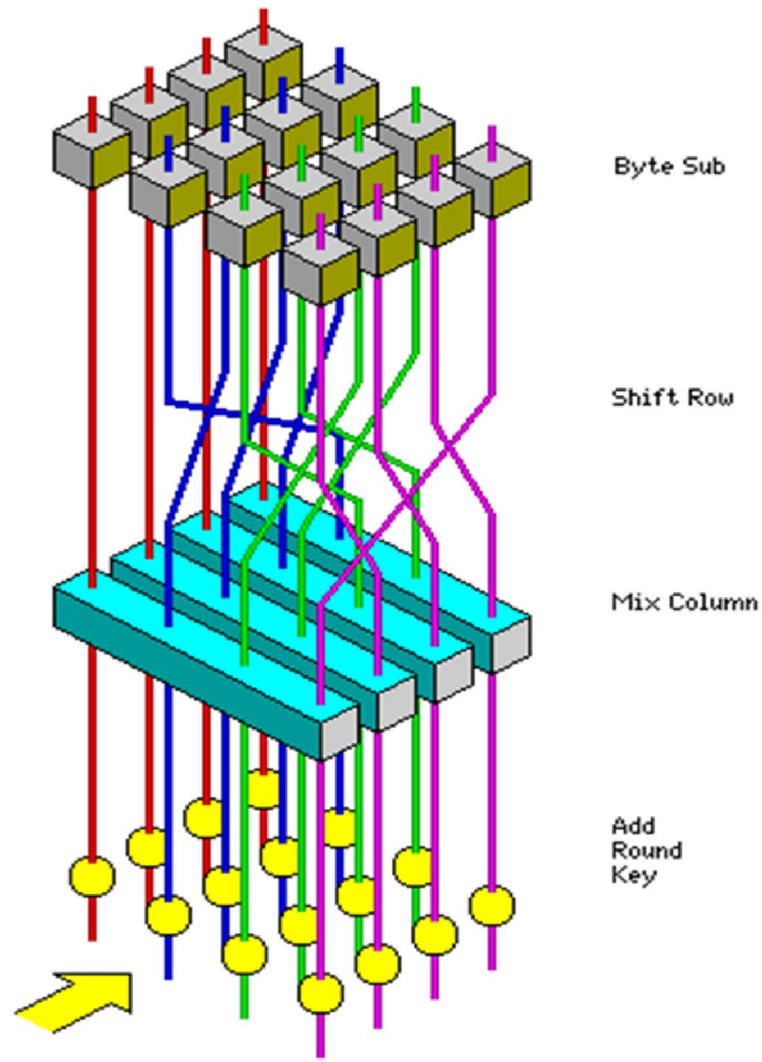
$$\begin{aligned}\{57\} \bullet \{13\} &= (\{57\} \bullet \{01\}) \oplus \underline{(\{57\} \bullet \{02\})} \oplus \underline{(\{57\} \bullet \{10\})} \\ &= \{57\} \oplus \{ae\} \oplus \{07\} \\ &= \{fe\}\end{aligned}$$

AES Encryption and Decryption



AES Round

1. Byte substitution using non-linear S-Box (independently on each byte)
2. Shift rows (square)
3. Mix columns – matrix multiplication by polynomial
4. XOR with round key



Convert to State Array

Input block:

0	4	8	12
1	5	9	13
2	6	10	14
3	7	11	15

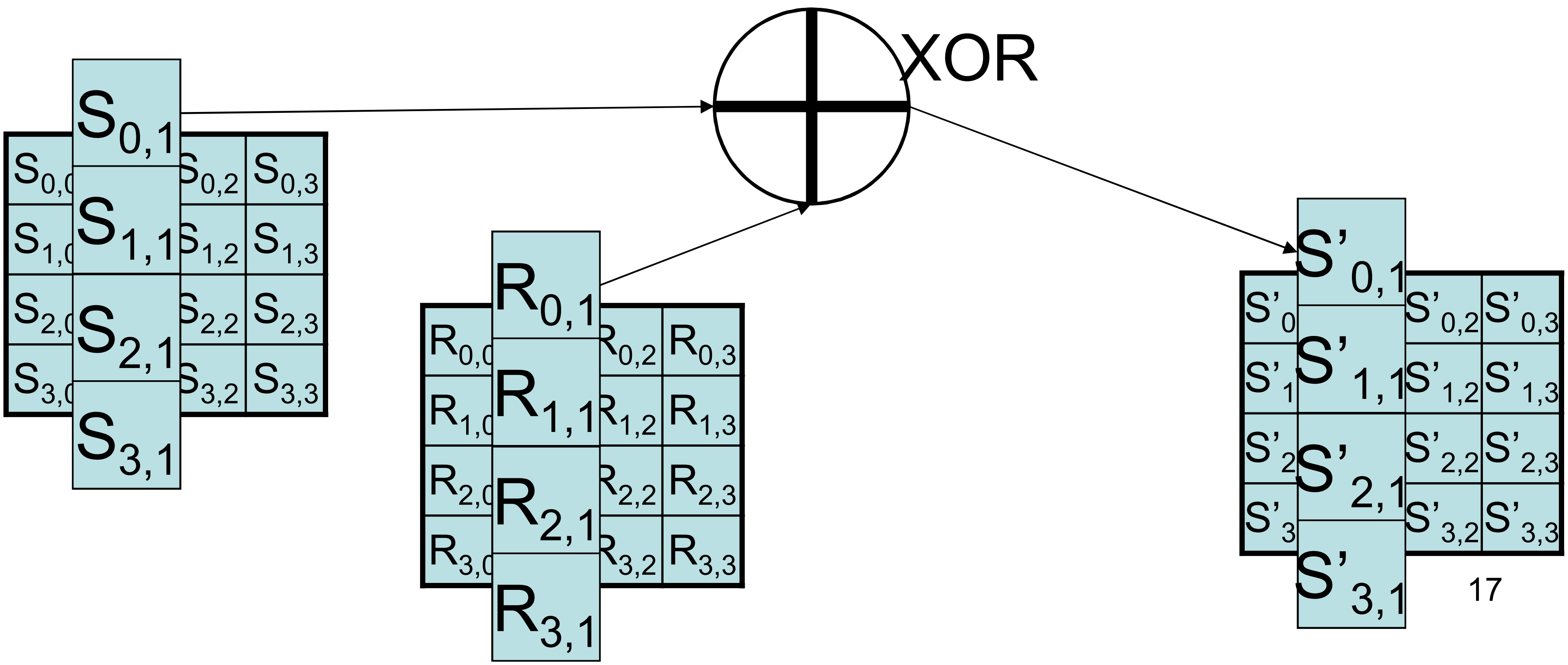
=

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

See The Rijndael Animation

AddRoundKey

- XOR each byte of the round key with its corresponding byte in the state array



SubBytes

- SubBytes is the SBOX for AES
- This make AES a non-linear cryptographic system.
- For every value of byte there is a unique value for byte’

input $S=\{3D\}_{16}$

1.Take multiplicative inverse in $GF(2^8)$ $x^8 + x^4 + x^3 + x + 1$

: $S \rightarrow S^{-1}$

2.Apply affine transformation over GF(2) as follows:
$S' = M \cdot S^{-1} + C$ $(C=\{27\}_{16})$

– where S and S’ are input/output bytes in 8-D vector formats
– It is faster to use a substitution table (and easier).

P99-understanding Cryptography

Table 4.2 Multiplicative inverse table in $GF(2^8)$ for bytes xy used within the AES S-Box

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

Affine transform

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

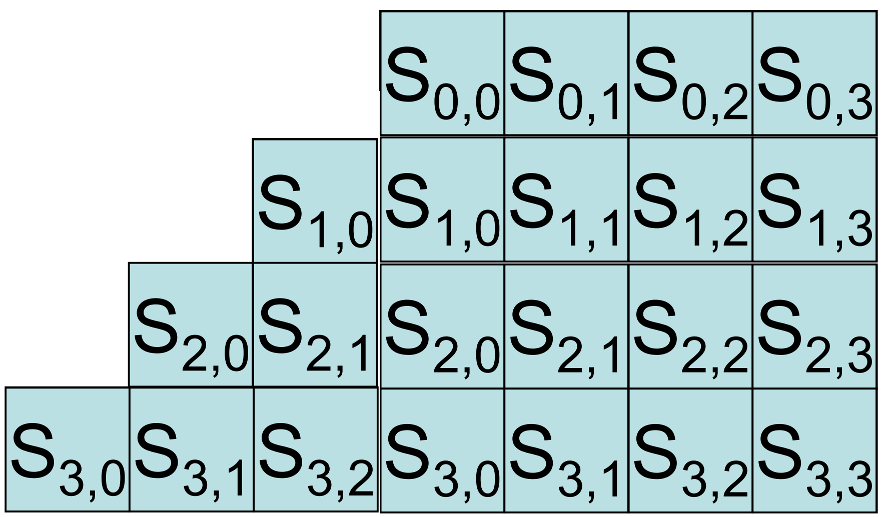
SubBytes

- Replace each byte in the state array with its corresponding value from the S-Box

		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

ShiftRows

- Last three rows are cyclically shifted



ShiftRows

- Simple routine which performs a left shift rows 1, 2 and 3 by 1, 2 and 3 bytes respectively This with shift rows provides diffusion

Before Shift Rows

53	CA	70	0C
D0	B7	D6	DC
51	04	F8	32
63	BA	68	79

→

After Shift Rows

53	CA	70	0C
B7	D6	DC	D0
F8	32	51	04
79	63	BA	68

MixColumns

- Apply MixColumn transformation to each column
- The columns are considered polynomials over $GF(2^8)$ and multiplied modulo x^4+1 with $a(x)$
- where $a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$
- This can also be written as matrix multiplication.

MixColumns

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} \left. \vphantom{\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}} \right\} \quad \begin{aligned} d_0 &= 2b_0 + 3b_1 + b_2 + b_3 \\ d_1 &= b_0 + 2b_1 + 3b_2 + b_3 \\ d_2 &= b_0 + b_1 + 2b_2 + 3b_3 \\ d_3 &= 3b_0 + b_1 + b_2 + 2b_3 \end{aligned}$$

Multiplication by 2 in $GF(2^8)$ takes some work:
If multiplying by a value < 0x80 just shift all the bits left by 1
If multiplying by a value ≥ 0x80 shift left by 1 and XOR with 0x1b

To Multiply by 3 in $GF(2^8)$:

a * 0x03 = a * (0x02 + 0x01) = (a * 0x02) (a * 0x01)

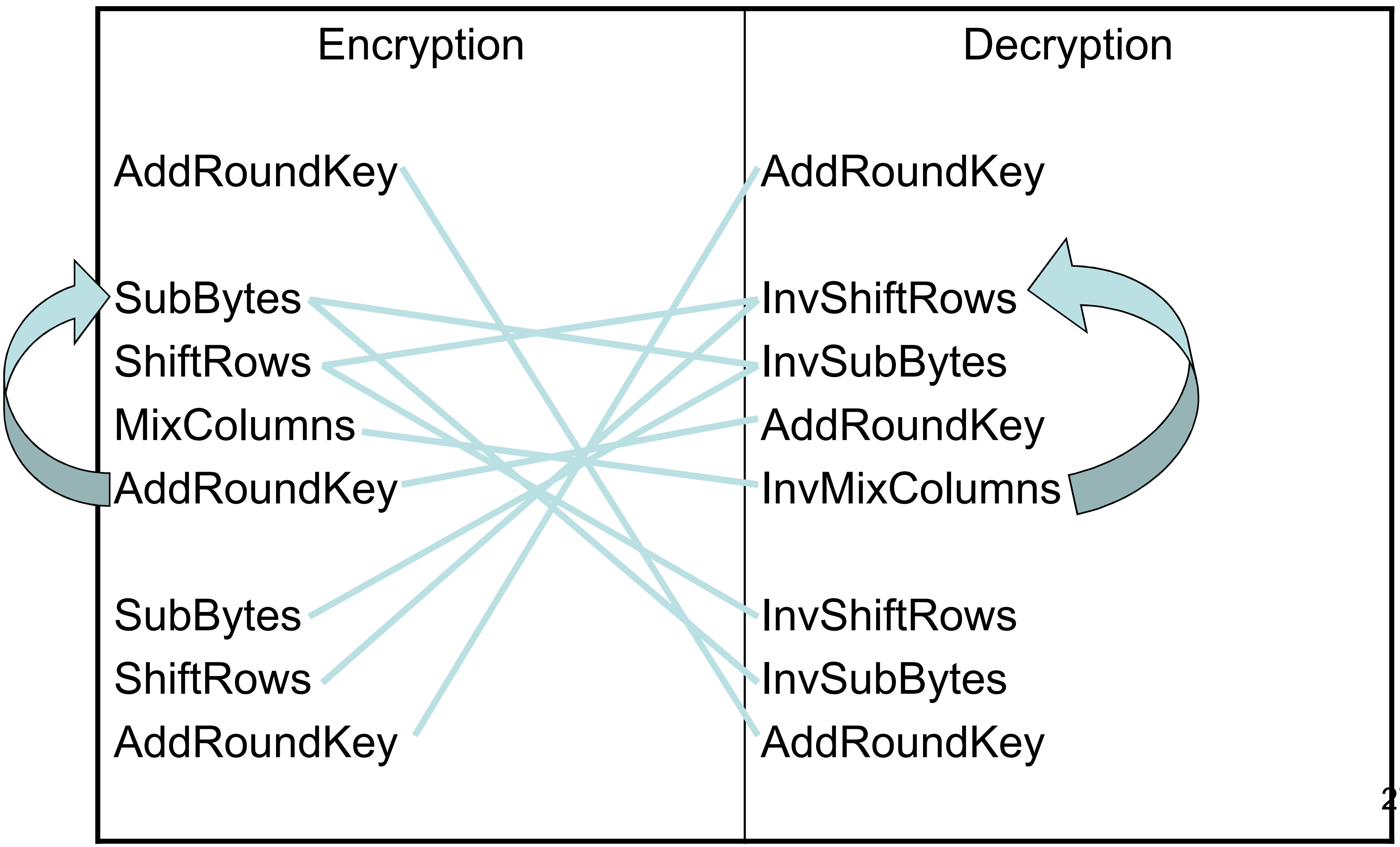
Lab AES Mix-column by python

1.สร้าง function คูณภายใต้ GF(2^8)

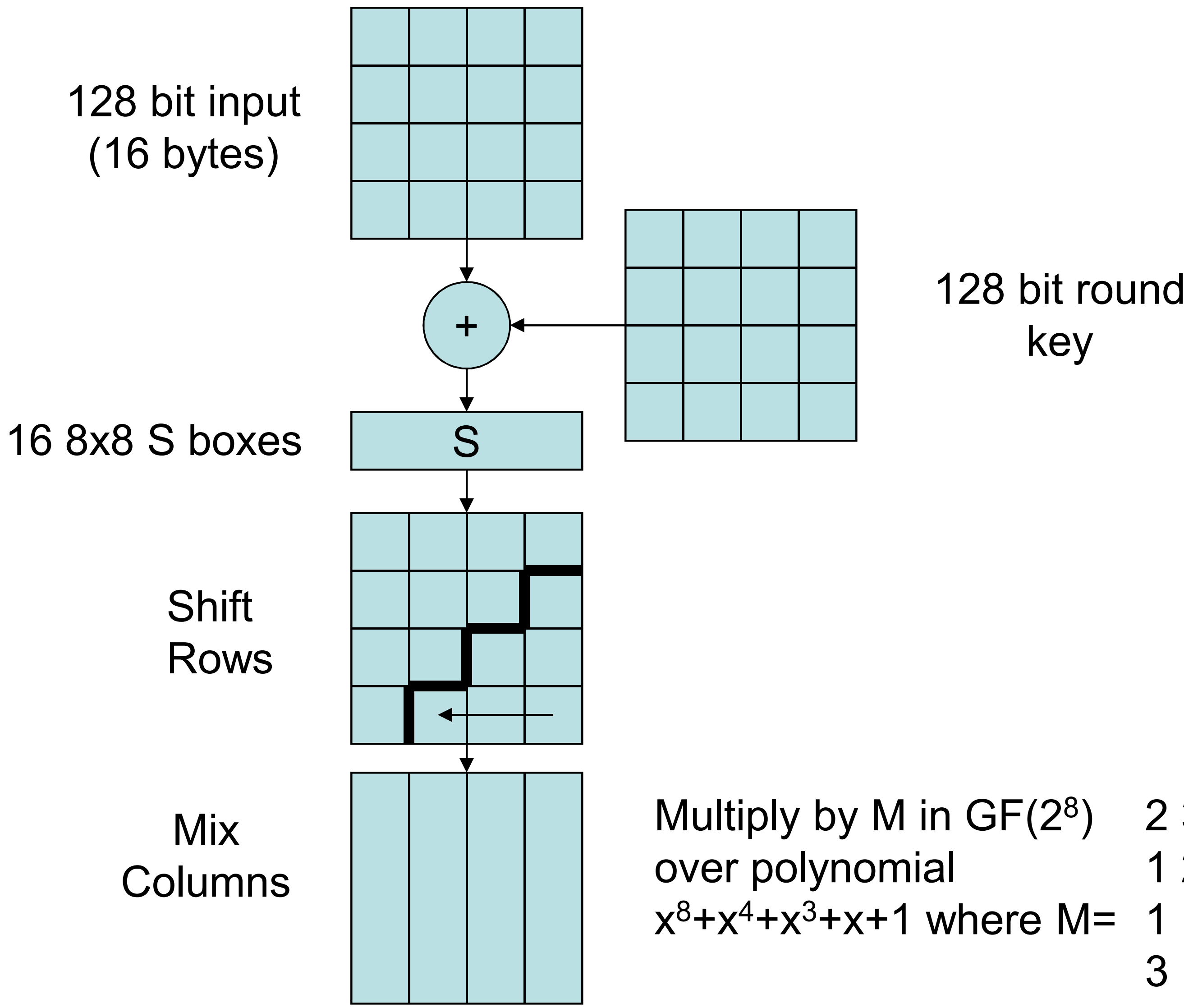
2.สร้าง function คูณ matrix

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

Encrypt and Decrypt



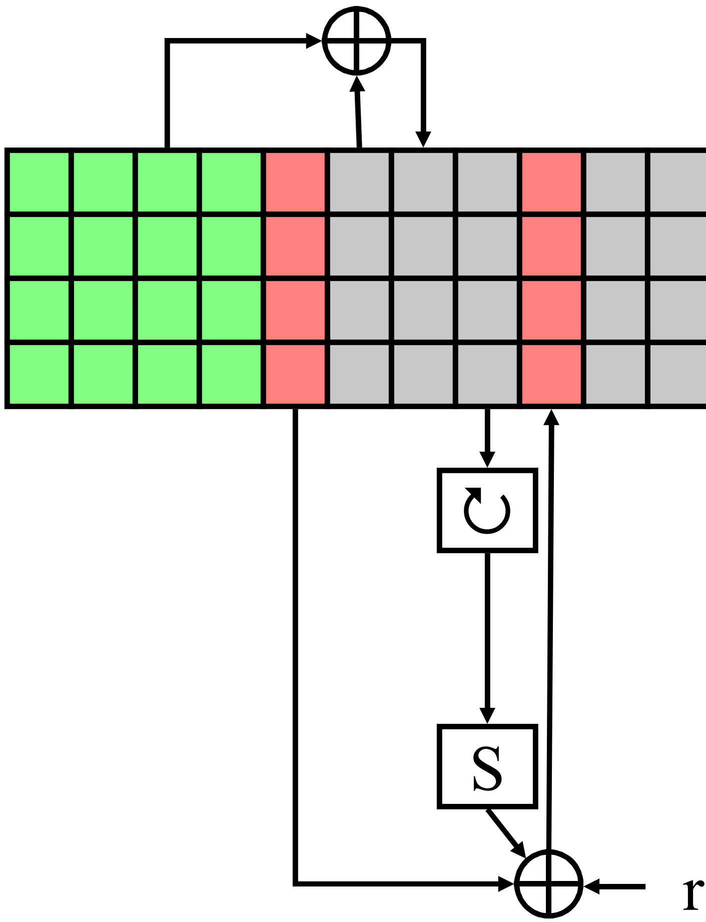
AES Round



Multiply by M in $GF(2^8)$ over polynomial $x^8+x^4+x^3+x+1$ where M=

$$M = \begin{matrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{matrix}$$

Key schedule (128 bits)

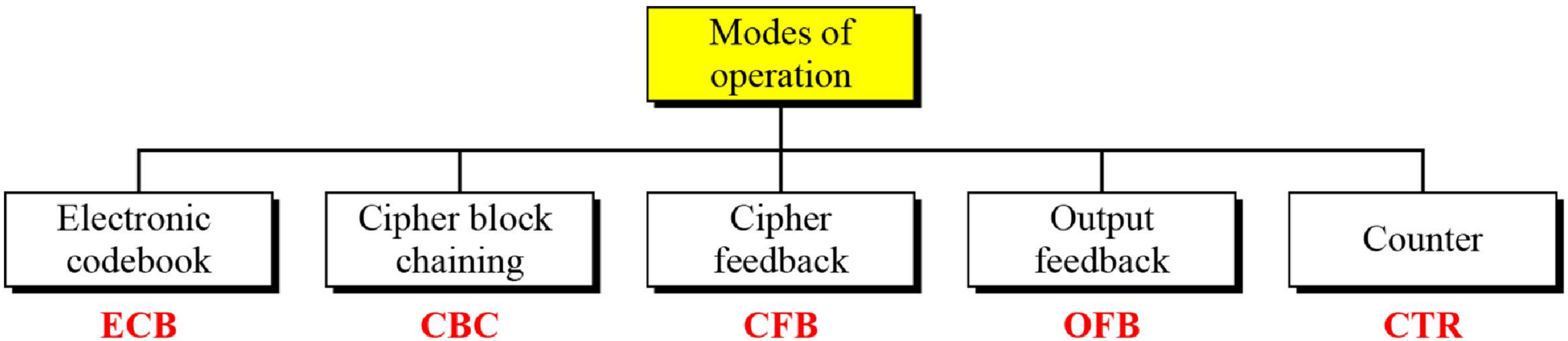


AES-mode

- **Overview of Modes of Operation**
- EBC, CBC, CTR
- Notes and Remarks on each modes

Modes of Operation Taxonomy

- Current well-known modes of operation



Mode Technical Notes

- Initialize Vector (IV)
  - a block of bits to randomize the encryption and hence to produce distinct ciphertext
- Nonce : Number (used) Once
  - Random of psuedorandom number to ensure that past communications can not be reused in replay attacks
  - Some also refer to initialize vector as nonce
- Padding
  - final block may require a padding to fit a block size
  - Method
    - Add null Bytes
    - Add 0x80 and many 0x00
    - Add the *n* bytes with value *n*

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

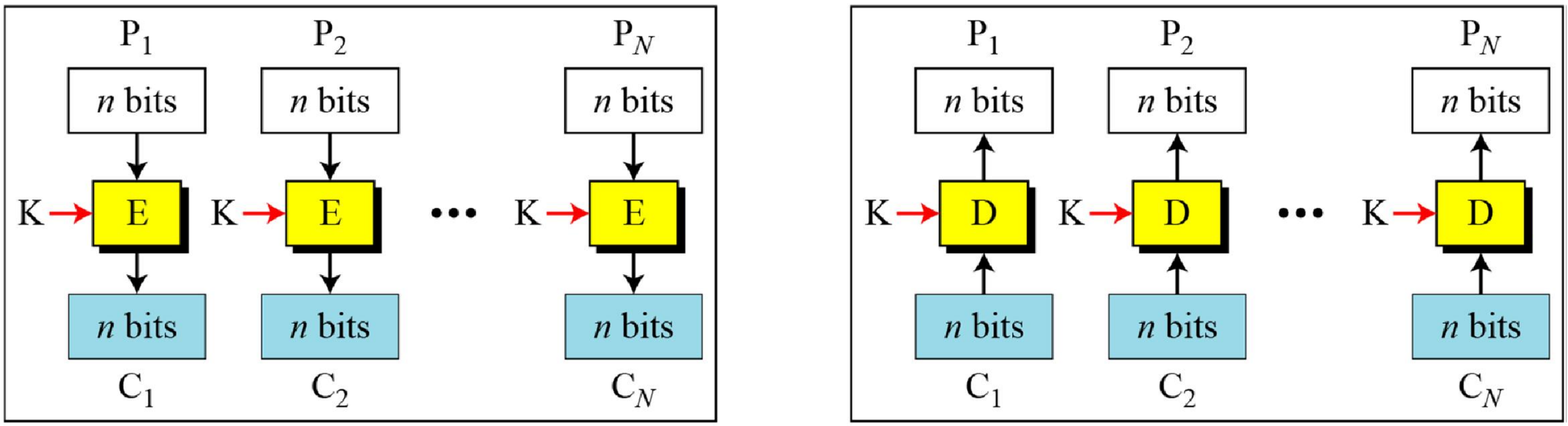
$$C_i = E_K (P_i)$$

- Uses: secure transmission of single values

ECB Scheme

Encryption: $C_i = E_K (P_i)$ Decryption: $P_i = D_K (C_i)$

E: Encryption D: Decryption
P_i: Plaintext block i C_i: Ciphertext block i
K: Secret key



Encryption Decryption

Remarks on ECB

- Strength: it's simple.

- Weakness:
  – Repetitive information contained in the plaintext may show in the ciphertext, if aligned with blocks.
  – If the same message is encrypted (with the same key) and sent twice, their ciphertext are the same.
- Typical application:
  – secure transmission of short pieces of information (e.g. a temporary encryption key)

Lab 1 ECB_AES

```
from Crypto.Cipher import AES
from binascii import hexlify
k1='abcdefghijklmnop'
print('k1=',k1)
p='00000000000000000000000000000000'
print('plain_text',p)
cipher = AES.new(k1,AES.MODE_ECB)
c =cipher.encrypt(p)
print('cipher=',hexlify(c))
decipher = AES.new(k1, AES.MODE_ECB)
msg=decipher.decrypt(c)
print('decrypt=',msg)
```

Lab 2 ECB_AES/rand key

```
from Crypto.Cipher import AES
from binascii import hexlify
from os import urandom
k2=urandom(16)
print('k2=',hexlify(k2))
p='00000000000000000000000000000000'
cipher = AES.new(k2,AES.MODE_ECB)
c =cipher.encrypt(p)
print('c=',hexlify(c))
decipher = AES.new(k2, AES.MODE_ECB)
msg=decipher.decrypt(c)
print('p=',msg)
```

Cipher Block Chaining (CBC)

- Solve security deficiencies in ECB
  - Repeated same plaintext block result different ciphertext block
- Each previous cipher blocks is chained to be input with current plaintext block, hence name
- Use Initial Vector (IV) to start process

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

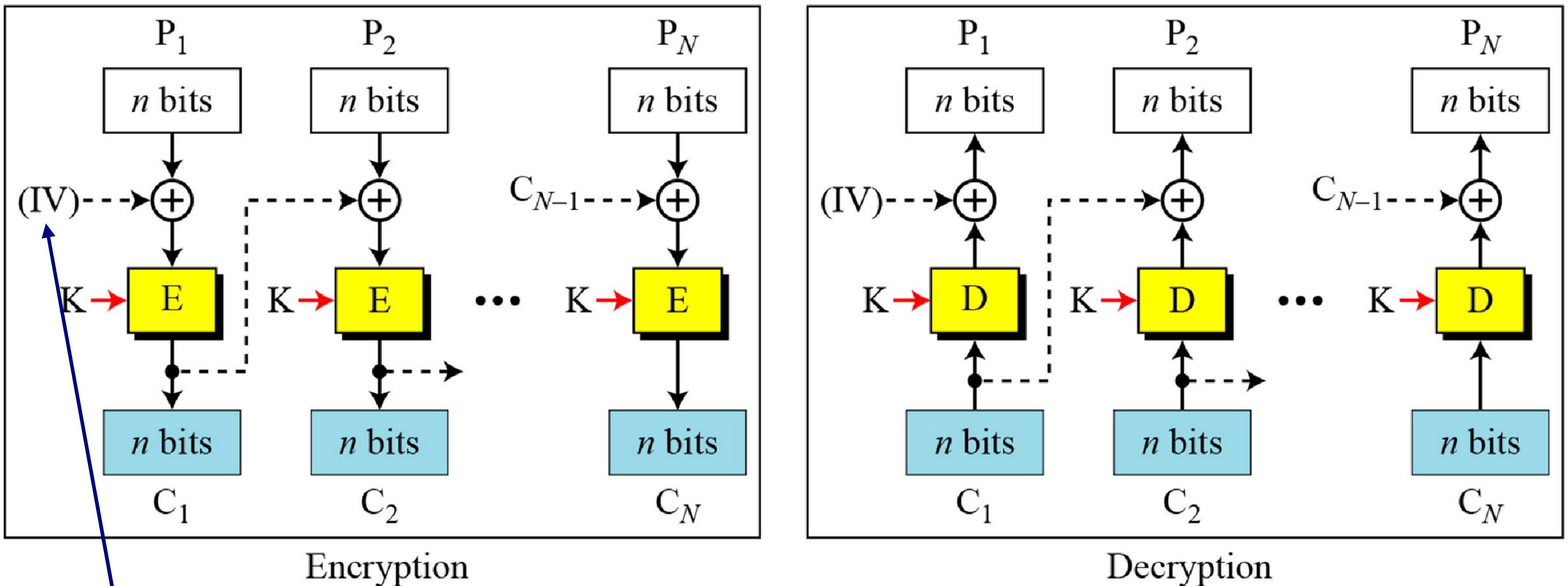
$$C_0 = IV$$

- Uses: bulk data encryption, authentication

CBC scheme

E: Encryption
P_i: Plaintext block i
K: Secret key

D : Decryption
C_i : Ciphertext block i
IV: Initial vector (C_0)



Encryption:	**Decryption:**
$C_0 = IV$	$C_0 = IV$
$C_i = E_K(P_i \oplus C_{i-1})$	$P_i = D_K(C_i) \oplus C_{i-1}$

Remarks on CBC

- The encryption of a block depends on the current and **all** blocks before it.

- So, repeated plaintext blocks are encrypted differently.

- Initialization Vector (IV)
  - May sent encrypted in ECB mode before the rest of ciphertext

LAB 3 AES CBC

```
from Crypto.Cipher import AES
from binascii import hexlify
from os import urandom
k1='abcdefghijklmnop'
iv=urandom(16)
iv0='0000000000000000'
print('k1=',k1)
print('iv=',hexlify(iv))
p='00000000000000000000000000000000'
cipher = AES.new(k1,AES.MODE_CBC,iv0)
c =cipher.encrypt(p)
print('c=',hexlify(c))
decipher = AES.new(k1, AES.MODE_CBC,iv0)
msg=decipher.decrypt(c)
print('p=',msg)
```

Counter (CTR)

- Encrypts counter value with the key rather than any feedback value (no feedback)
- Counter for each plaintext will be different
  - can be any function which produces a sequence which is guaranteed not to repeat for a long time
- Relation

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

$$O_i = E_K(i)$$

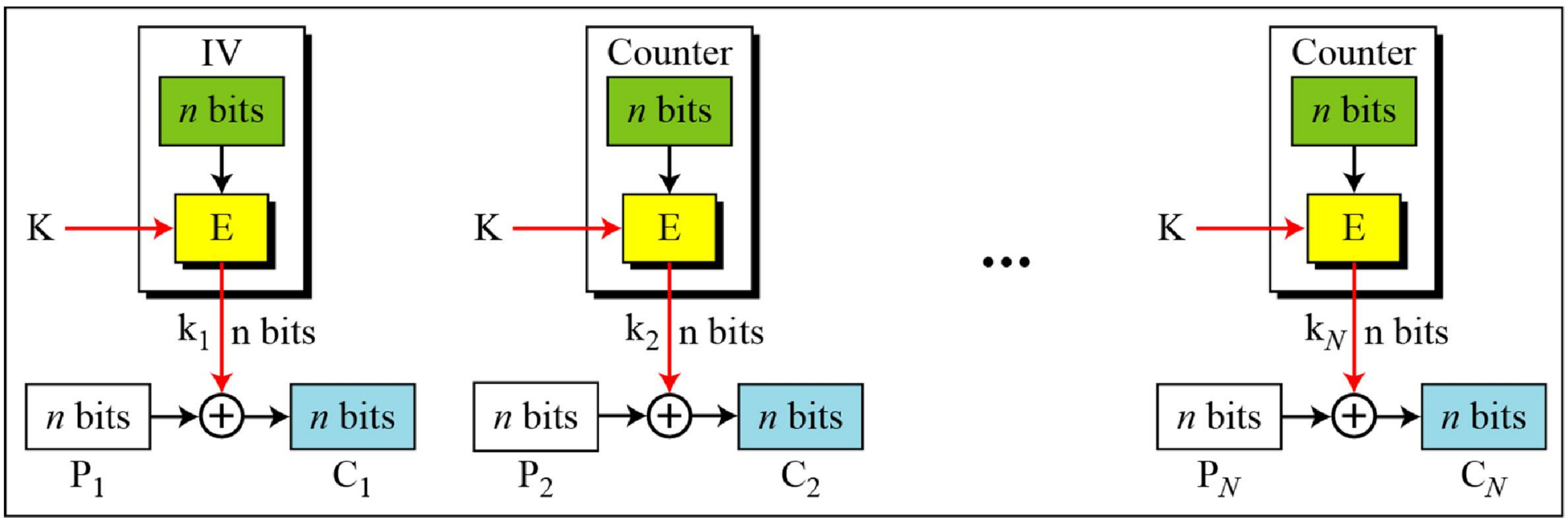
- Uses: high-speed network encryptions

CTR Scheme

E : Encryption
P_i : Plaintext block i
K : Secret key

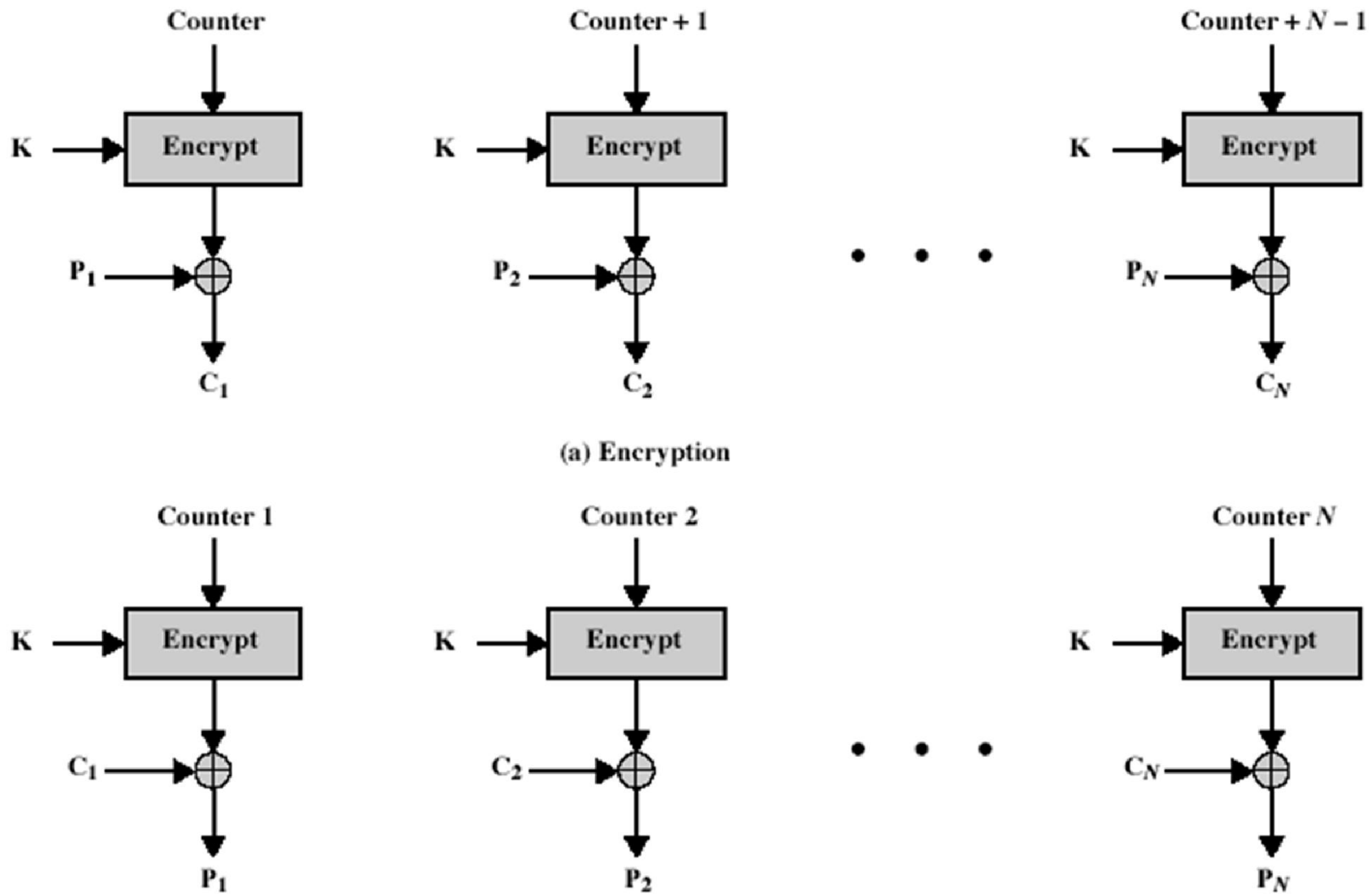
IV: Initialization vector
C_i : Ciphertext block i
k_i : Encryption key i

The counter is incremented for each block.



Encryption

CTR Encryption and Decryption



(a) Encryption

(b) Decryption

Lab 4 AES CTR

```
from Crypto.Cipher import AES
from binascii import hexlify
from os import urandom
from Crypto.Util import Counter
from struct import unpack
```

```
k2=urandom(16)
iv=urandom(8)
nonce=unpack('<Q',urandom(8))[0]
print('k2=',hexlify(k2))
p='1234'
ctr=Counter.new(128,initial_value=nonce)
cipher = AES.new(k2,AES.MODE_CTR,counter=ctr)
c =cipher.encrypt(p)
print('cipher=',hexlify(c))
ctr=Counter.new(128,initial_value=nonce)
cipher = AES.new(k2,AES.MODE_CTR,counter=ctr)
msg =cipher.encrypt(c)
print('decrypt=',msg)LAb
```

Modes and IV

- An IV has different security requirements than a key
- Generally, an IV will not be reused under the same key
- CBC
  - reusing an IV leaks some information about the first block of plaintext, and about any common prefix shared by the two messages
- CTR
  - reusing an IV completely destroys security

CBC and CTR comparison

CBC	CTR
Padding needed	No padding
No parallel processing	Parallel processing
Separate encryption and decryption functions	Encryption function alone is enough
Random IV or a nonce	Unique nonce
Nonce reuse leaks some information about initial plaintext block	Nonce reuse will leak information about the entire message

Comparison of Modes

Mode	Description	Application
ECB	64-bit plaintext block encoded separately	Secure transmission of encryption key
CBC	64-bit plaintext blocks are XORed with preceding 64-bit ciphertext	Commonly used method. Used for authentication
CTR	Key calculated using the nonce and the counter value. Counter is incremented for each block	General purpose block oriented transmission. Used for high-speed communications