Week 5: Modern Symmetric Ciphers (P2)

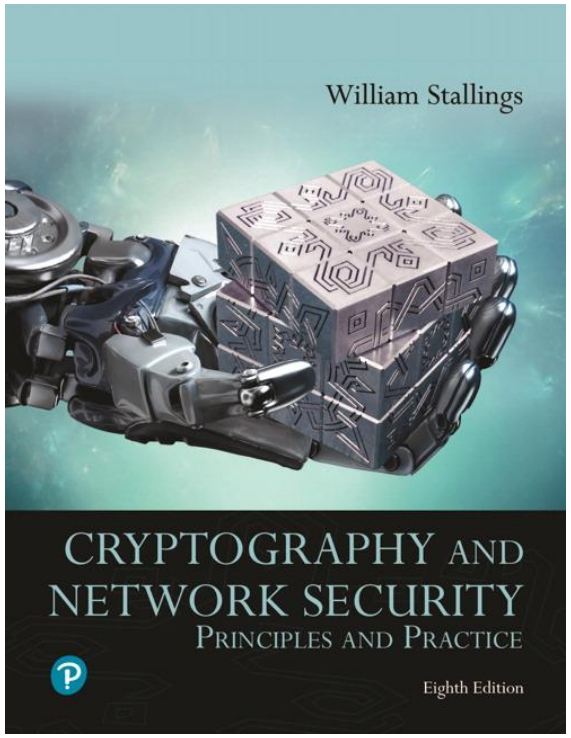
PhD. Ngoc-Tu Nguyen
tunn@uit.edu.vn

Outline

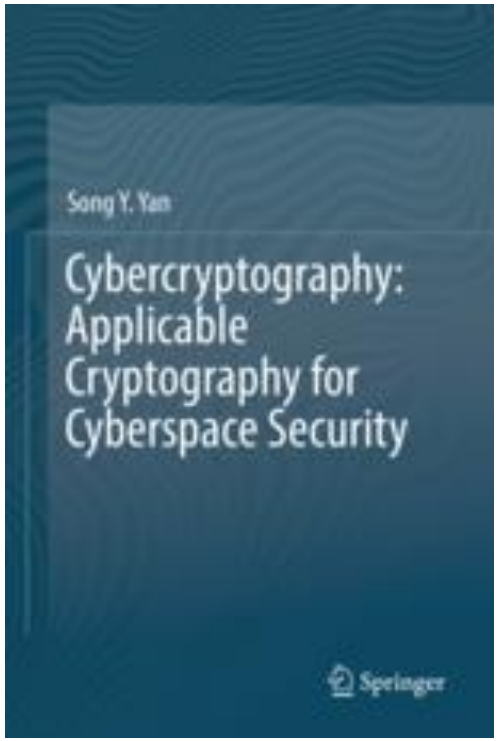
- Stream Cipher
- Block cipher
  - Data Encryption Standard (DES)
  - Advanced Encryption Standard (AES)
  - Mode of operation
  - Some other ciphers
    - Searchable encryption

Textbooks and References

- Text books

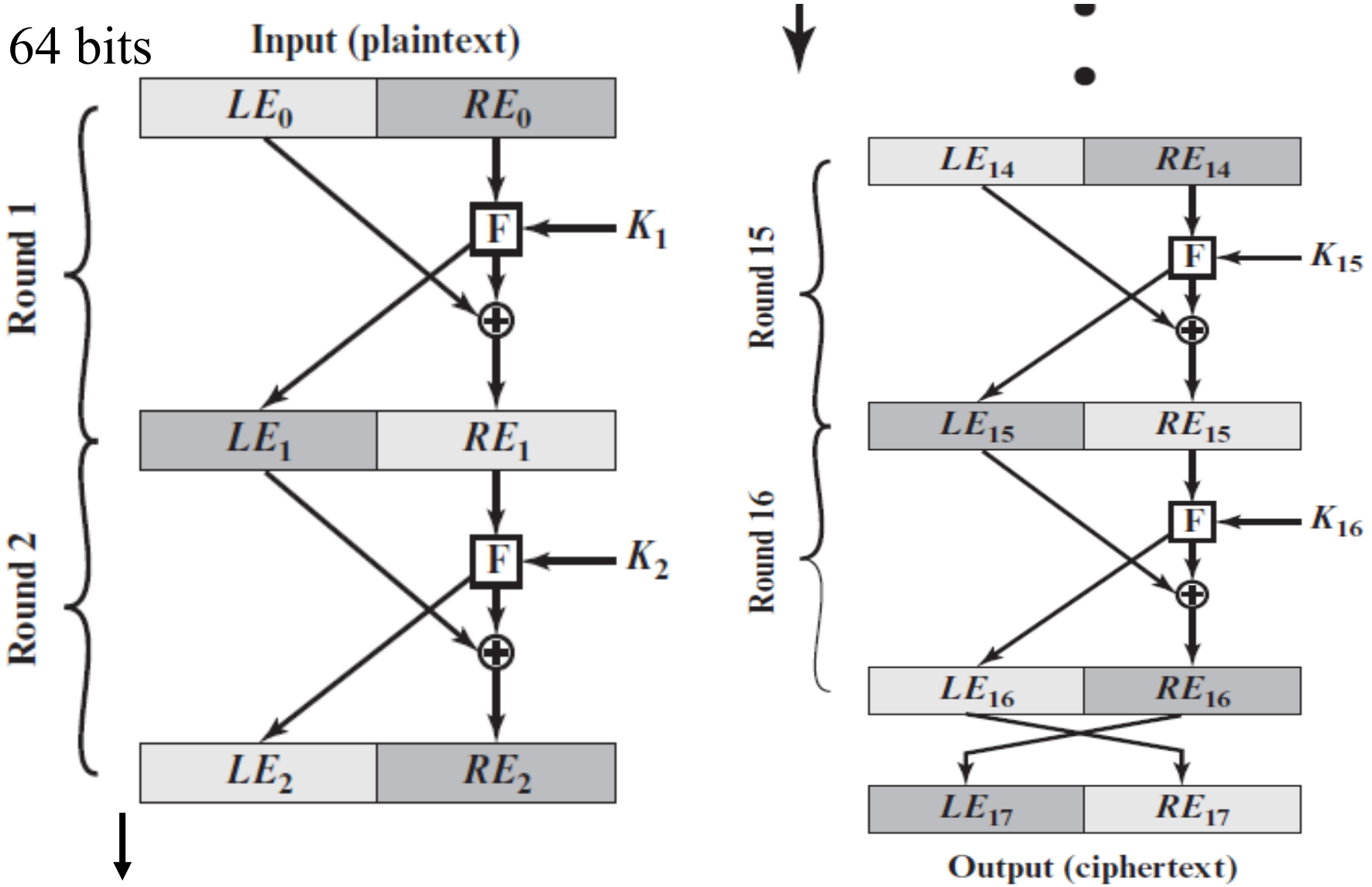


[1] Chapter 4,6



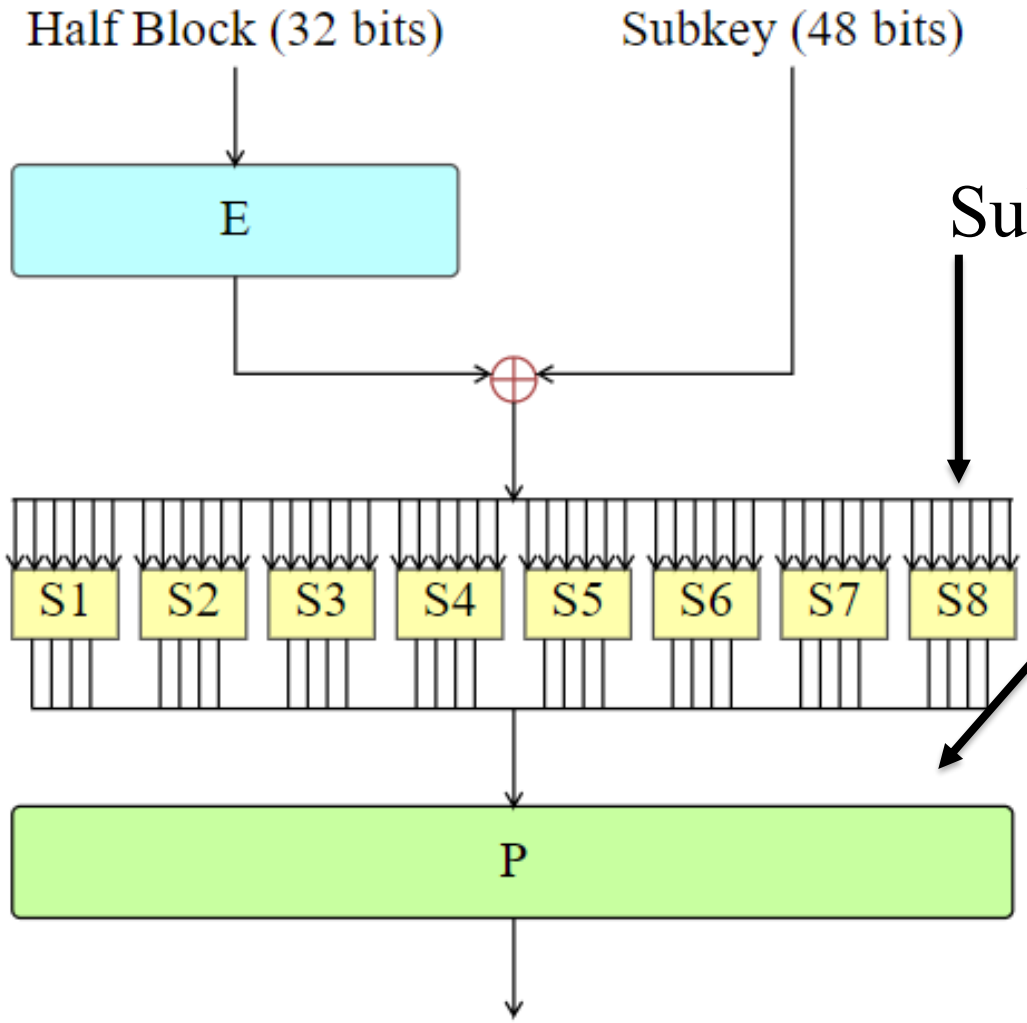
[2] Chapter 5

DES review


64 bits
Input (plaintext)
LE_0
RE_0
F
K_1
LE_1
RE_1
F
K_2
LE_2
RE_2
Round 1
Round 2
LE_{14}
RE_{14}
F
K_{15}
LE_{15}
RE_{15}
F
K_{16}
LE_{16}
RE_{16}
LE_{17}
RE_{17}
Round 15
Round 16
Output (ciphertext)

DES review

$$F(R_{i-1}, K_i) = P(S(EP(R_{i-1}) \oplus K_i)), i = 1,\ldots,16$$



Substitution

Permutation

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 review

Security analysis

Key spaces: $\{0,1\}^{56} = 2^{56}$ possible keys

Brute Force attacks

1997	The DESCHALL Project breaks a message encrypted with DES for the first time in public.
1998	The EFF's DES cracker (Deep Crack) breaks a DES key in 56 hours.
1999	Together, Deep Crack and distributed.net break a DES key in 22 hours and 15 minutes.

2016	The Open Source password cracking software hashcat added in DES brute force searching on general purpose GPUs. Benchmarking shows a single off the shelf Nvidia GeForce GTX 1080 Ti GPU costing $1000 USD recovers a key in an average of 15 days (full exhaustive search taking 30 days). Systems have been built with eight GTX 1080 Ti GPUs which can recover a key in an average of under 2 days.[25]
2017	A chosen-plaintext attack utilizing a rainbow table can recover the DES key for a single specific chosen plaintext *1122334455667788* in 25 seconds. A new rainbow table has to be calculated per plaintext. A limited set of rainbow tables have been made available for download.[26]

https://en.wikipedia.org/wiki/Data_Encryption_Standard

Outline

- Stream Cipher
- Block cipher
  - Data Encryption Standard (DES)
  - Advanced Encryption Standard (AES)
  - Some other ciphers
    - Searchable encryption

Advanced Encryption Standard

- ✓ Advanced Encryption Standard competition began in 1997
- ✓ Rijndael was selected to be the new AES in 2001
- ✓ AES basic structures:
  - block cipher, but not Feistel cipher
  - encryption and decryption are similar, but not symmetrical
  - basic unit: byte, not bit
  - block size: 16-bytes (128 bits)
  - three different key lengths: 128, 192, 256 bits (AES-128, AES-192, AES-256)
  - each 16-byte block is represented as a 4 x 4 square matrix, called the ***state matrix***
  - the number of rounds depends on key lengths
  - 4 simple operations on the state matrix every round (except the last round)

Finite Field Arithmetic

- p is a prime number

  $Z_p = \{0,1,2,...,p-1\}$ is a finite field;

- $N=2^n$ is a composite number

  $Z_{2^n} = \{0,1,2,...,2^n-1\}$ is a finite field?

  NO!

  Example: $Z_{2^3} = \{0,1,2,3,4,5,6,7\}$?

mode 8	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7
2	2	4	6	0	2	4	6
3	3	6	1	4	7	2	5
4	4	0	4	0	4	0	4
5	5	2	7	4	1	6	3
6	6	4	2	0	6	4	2
7	7	6	5	4	3	2	1

$3.3 \bmod 8 = 1$ $\quad$ $2.x \bmod 8 = 1$?

$5.5 \bmod 8 = 1$ $\quad$ $4.x \bmod 8 = 1$?

$7.7 \bmod 8 = 1$ $\quad$ $6.x \bmod 8 = 1$?

But $GF(Z_{2^n})$ is a finite field!

Finite Field Arithmetic

- If one of the operations used in the algorithm is division, then we need to work in arithmetic defined over a field
  - Division requires: nonzero element have a multiplicative inverse
- For convenience and for implementation efficiency we would like to work with integers that fit exactly into a given number of bits with no wasted bit patterns
  - Integers in the range 0 through $2^n - 1$, which fit into an *n*-bit word
- The set of such integers, Z_{2^n}, using modular arithmetic, **is not a field**
  - The integer 2 has no multiplicative inverse in Z_{2^n}, (no integer *b,* such that:

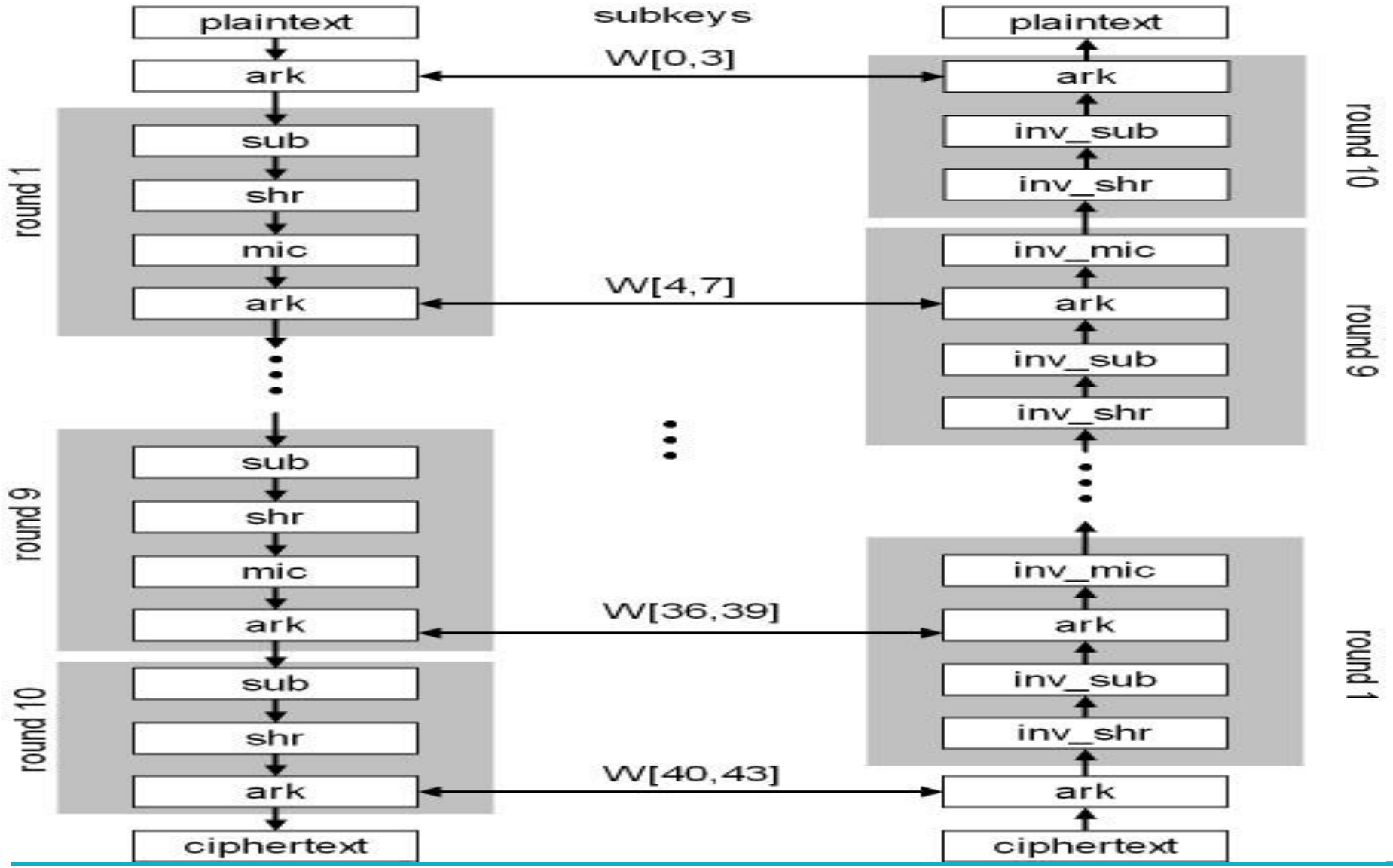
$$2.b \bmod 2^n = 1)$$

- A finite field containing 2^n elements is referred to as $GF(2^n)$
  - Every polynomial in $GF(2^n)$ can be represented by an n-bit number

The Four Simple Operations

- **substitute-bytes** (sub)
  - Non-linear operation based on a defined **substitution box**
  - Used to resist cryptanalysis and other mathematical attacks
- **shift-rows** (shr)
  - Linear operation for producing **diffusion**
- **mix-columns** (mic)
  - Elementary operation also for producing **diffusion**
- **add-round-key** (ark)
  - Simple set of $\oplus$ operations on state matrices
  - Linear operation
  - Produces **confusion**

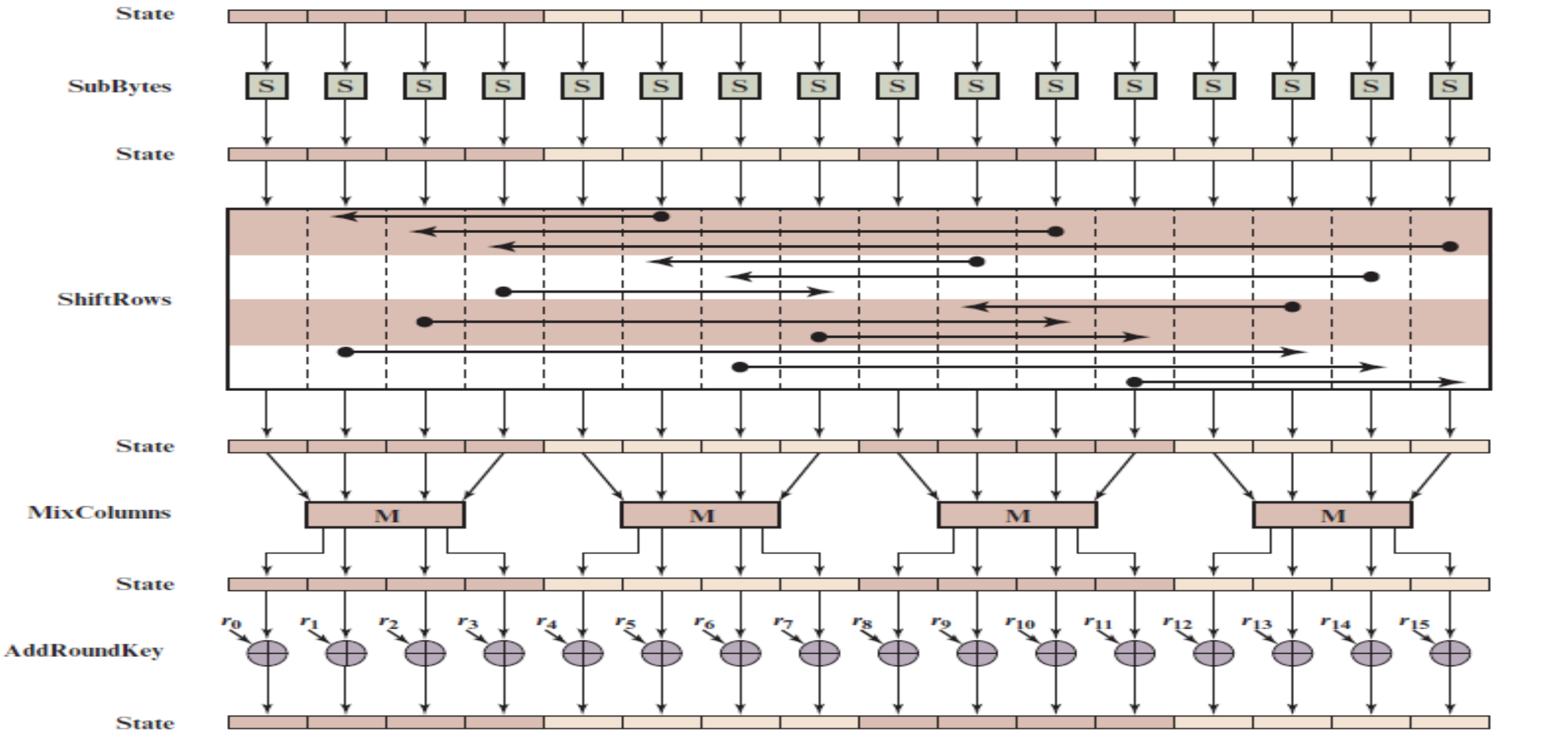
AES-128



AES-128: 10 rounds
AES-192: 12 rounds
AES-256:14 round

AES Encryption Round

16 bytes



AES Substution Box (S-Box)

- S-box: a 16×16 matrix built from operations over finite field GF(2^8)
  - permute all 256 elements in GF(2^8)
  - each element and its index are represented by two hexadecimal digits
- Let $w = b_0 \ldots b_7$ be a byte. Define a byte-substitution function S as follows:

  Let $i = b_0 b_1 b_2 b_3$, the binary representation of the row index

  Let $j = b_4 b_5 b_6 b_7$, the binary representation of the column index

  Let $S(w) = s_{ij,}$ $S^{-1}(w) = s'_{ij}$
- We have $S(S^{-1}(w)) = w$ and $S^{-1}(S(w)) = w$

AES S-Boxes (1 of 2)

x \ y	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

(a) S-box

P=1101 0011

↓13 ↓

i =D j= 3

⇩

C=66 = $(01100110)_2$

AES inverse substution Box (2 of 2)

x \ 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
7	D0	2C	1E	8F	CA	3F	0F	02	C1	AF	BD	03	01	13	8A	6B
8	3A	91	11	41	4F	67	DC	EA	97	F2	CF	CE	F0	B4	E6	73
9	96	AC	74	22	E7	AD	35	85	E2	F9	37	E8	1C	75	DF	6E
A	47	F1	1A	71	1D	29	C5	89	6F	B7	62	0E	AA	18	BE	1B
B	FC	56	3E	4B	C6	D2	79	20	9A	DB	C0	FE	78	CD	5A	F4
C	1F	DD	A8	33	88	07	C7	31	B1	12	10	59	27	80	EC	5F
D	60	51	7F	A9	19	B5	4A	0D	2D	E5	7A	9F	93	C9	9C	EF
E	A0	E0	3B	4D	AE	2A	F5	B0	C8	EB	BB	3C	83	53	99	61
F	17	2B	04	7E	BA	77	D6	26	E1	69	14	63	55	21	0C	7D

(b) Inverse S-box

$C=66 = (0110\ 0110)_2$

i =6 j=6

$P=D3 = (1101\ 0011)_2$

Substitute-Bytes (*sub*)

- Substitution function that takes a byte as an input, uses its first four bits as the row index and the last four bits as the column index, and outputs a byte using a table-lookup at the S-box
- Let A be a state matrix. Then

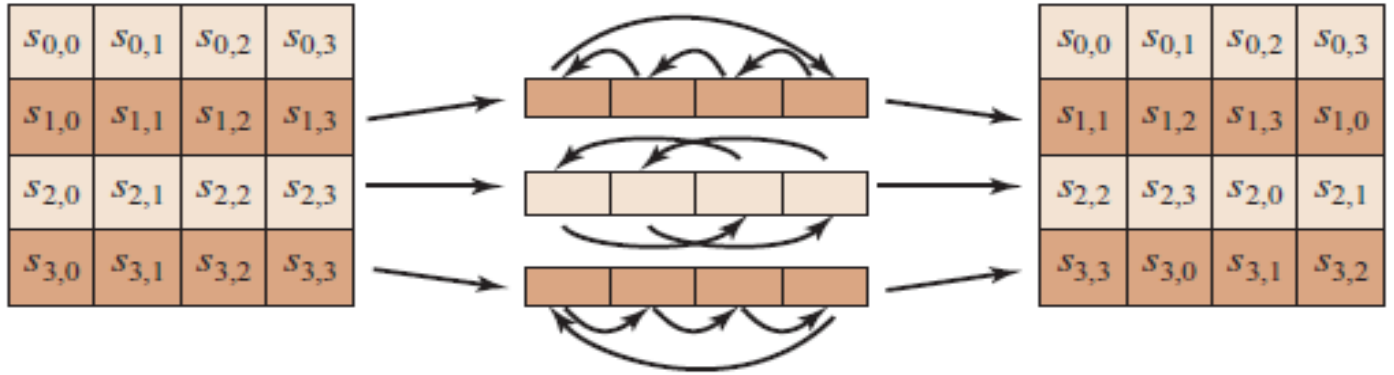
$$\begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix} \xrightarrow{\text{S-Box}} C = sub(A) = \begin{pmatrix} S(a_{0,0}) & S(a_{0,1}) & S(a_{0,2}) & S(a_{0,3}) \\ S(a_{1,0}) & S(a_{1,1}) & S(a_{1,2}) & S(a_{1,3}) \\ S(a_{2,0}) & S(a_{2,1}) & S(a_{2,2}) & S(a_{2,3}) \\ S(a_{3,0}) & S(a_{3,1}) & S(a_{3,2}) & S(a_{3,3}) \end{pmatrix}$$

- $sub^{-1}(A)$ will just be the inverse substitution operation applied to the matrix

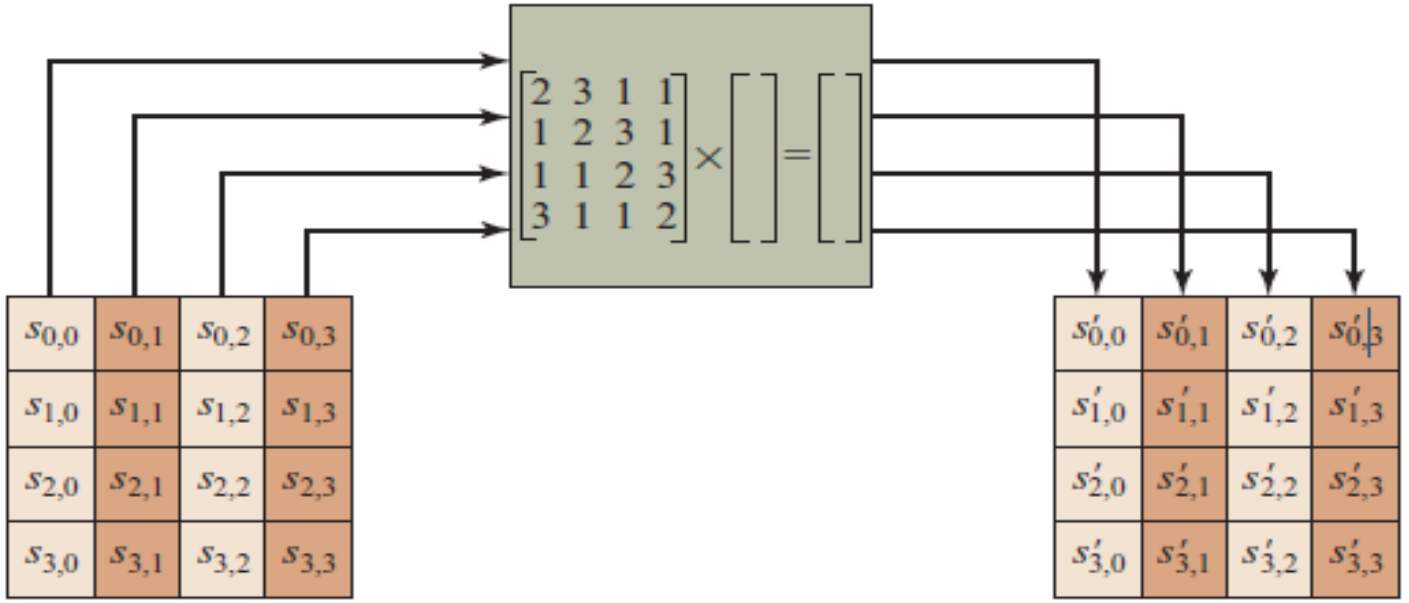
$$sub^{-1}(C) = \begin{pmatrix} S^{-1}(c_{0,0}) & S^{-1}(c_{0,1}) & S^{-1}(c_{0,2}) & S^{-1}(c_{0,3}) \\ S^{-1}(c_{1,0}) & S^{-1}(c_{1,1}) & S^{-1}(c_{1,2}) & S^{-1}(c_{1,3}) \\ S^{-1}(c_{2,0}) & S^{-1}(c_{2,1}) & S^{-1}(c_{2,2}) & S^{-1}(c_{2,3}) \\ S^{-1}(c_{3,0}) & S^{-1}(c_{3,1}) & S^{-1}(c_{3,2}) & S^{-1}(c_{3,3}) \end{pmatrix} \downarrow \text{Inverse S-Box}$$

- We have $sub(sub^{-1}(A)) = sub^{-1}(sub(A)) = A$

AES Row and Column Operations



(a) Shift row transformation



(b) Mix column transformation

Shift-Rows (shr)

- $shr(A)$ performs a left-circular-shift $i-1$ times on the i-th row in the matrix A

$$\begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix} \begin{matrix} \leftarrow 0 \\ \leftarrow 1 \\ \leftarrow 2 \\ \leftarrow 3 \end{matrix} \Rightarrow shr(A) = \begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,1} & a_{1,2} & a_{1,3} & a_{1,0} \\ a_{2,2} & a_{2,3} & a_{2,0} & a_{2,1} \\ a_{3,3} & a_{3,0} & a_{3,1} & a_{3,2} \end{pmatrix} = C$$

- $shr^{-1}(A)$ performs a right-circular-shift $i-1$ times on the i-th row in the matrix A

$$shr^{-1}(C) = \begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix}$$

Mix-Columns (*mic*)

$$A=\begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix}$$

Encrypt ⟹

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

$$A=\begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix}=C$$

Decrypt ⟸

$$M^{-1}=\begin{bmatrix} 14 & 11 & 13 & 9 \\ 9 & 14 & 11 & 13 \\ 13 & 9 & 14 & 11 \\ 11 & 13 & 9 & 14 \end{bmatrix}$$

$$C=\begin{pmatrix} c_{0,0} & c_{0,1} & c_{0,2} & c_{0,3} \\ c_{1,0} & c_{1,1} & c_{1,2} & c_{1,3} \\ c_{2,0} & c_{2,1} & c_{2,2} & c_{2,3} \\ c_{3,0} & c_{3,1} & c_{3,2} & c_{3,3} \end{pmatrix}$$

Addition, subtraction, multiplication, and division on bytes?

Finite Field Arithmetic (3/3)

- In the Advanced Encryption Standard (AES) all operations are performed on 8-bit (1 byte);
- The arithmetic operations of addition, multiplication, and division are performed over the **finite field GF(2^8)**

- **Rijndael's finite field** $GF(Z_{2^8})$

$$r = b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0 \text{ (1 byte)}$$

$$r(x) = b_7 x^7 + b_6 x^6 + b_5 x^5 + b_4 x^4 + b_3 x^3 + b_2 x^2 + b_1 x^1 + b_0$$

- addition,
- subtraction,
- multiplication,
- division on polynomials

mod $(x^8 + x^4 + x^3 + x + 1)$

$$GF(Z_{2^8}) = \{\text{GF}(2)[x]/(x^8 + x^4 + x^3 + x + 1)\}$$ https://en.wikipedia.org/wiki/Finite_field_arithmetic

Mix-Columns (mic)

- $mic(A) = [a'_{ij}]_{4\times 4}$ is determined by the following operation (j = 0, 1, 2, 3):

$$a'_{0,j} = M(a_{0,j}) \oplus [M(a_{1,j}) \oplus a_{1,j}] \oplus a_{2,j} \oplus a_{3,j}$$
$$a'_{1,j} = a_{0,j} \oplus M(a_{1,j}) \oplus [M(a_{2,j}) \oplus a_{2,j}] \oplus a_{3,j}$$
$$a'_{2,j} = a_{0,j} \oplus a_{1,j} \oplus M(a_{2,j}) \oplus [M(a_{3,j}) \oplus a_{3,j}]$$
$$a'_{3,j} = [M(a_{0,j}) \oplus a_{0,j}] \oplus a_{1,j} \oplus a_{2,j} \oplus M(a_{3,j})$$

Mix-Columns (mic)

- $mic^{-1}(A)$ is defined as follows:
  - Let w be a byte and i a positive integer:

    $M^{i}(\mathrm{w}) = M\,(M^{i-1}(\mathrm{w}))\ (i > 1),\ M^{1}(\mathrm{w}) = M\,(\mathrm{w})$
  - Let

    $M_1(\mathrm{w}) = M^3(\mathrm{w}) \oplus M^2(\mathrm{w}) \oplus M(\mathrm{w})$

    $M_2(\mathrm{w}) = M^3(\mathrm{w}) \oplus M(\mathrm{w}) \oplus \mathrm{w}$

    $M_3(\mathrm{w}) = M^3(\mathrm{w}) \oplus M^2(\mathrm{w}) \oplus \mathrm{w}$

    $M_4(\mathrm{w}) = M^3(\mathrm{w}) \oplus \mathrm{w}$
- $mic^{-1}(A) = [\mathrm{a''}_{ij}]_{4\times 4}$:

  $\mathrm{a''}_{0,j} = M_1(\mathrm{a}_{0,j}) \oplus M_2(\mathrm{a}_{1,j}) \oplus M_3(\mathrm{a}_{2,j}) \oplus M_4(\mathrm{a}_{3,j})$

  $\mathrm{a''}_{1,j} = M_4(\mathrm{a}_{0,j}) \oplus M_1(\mathrm{a}_{1,j}) \oplus M_2(\mathrm{a}_{2,j}) \oplus M_3(\mathrm{a}_{3,j})$

  $\mathrm{a''}_{2,j} = M_3(\mathrm{a}_{0,j}) \oplus M_4(\mathrm{a}_{1,j}) \oplus M_1(\mathrm{a}_{2,j}) \oplus M_2(\mathrm{a}_{3,j})$

  $\mathrm{a''}_{3,j} = M_2(\mathrm{a}_{0,j}) \oplus M_3(\mathrm{a}_{1,j}) \oplus M_4(\mathrm{a}_{2,j}) \oplus M_1(\mathrm{a}_{3,j})$
- We have $mic(mic^{-1}(A)) = mic^{-1}(mic(A)) = A$

Add Round Keys (*ark*)

- Rewrite K_i as a 4 x 4 matrix of bytes:

$$K_i = \begin{pmatrix} k_{0,0} & k_{0,1} & k_{0,2} & k_{0,3} \\ k_{1,0} & k_{1,1} & k_{1,2} & k_{1,3} \\ k_{2,0} & k_{2,1} & k_{2,2} & k_{2,3} \\ k_{3,0} & k_{3,1} & k_{3,2} & k_{3,3} \end{pmatrix} \quad A = \begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix}$$

  where each element is a byte and $W[4i + j] = k_{0,j} k_{1,j} k_{2,j} k_{3,j}, j = 0, 1, 2, 3$

- Initially, let $A = M$

$$ark(A, K_i) = A \oplus K_i = \begin{pmatrix} k_{0,0} \oplus a_{0,0} & k_{0,1} \oplus a_{0,1} & k_{0,3} \oplus a_{0,3} & k_{0,4} \oplus a_{0,4} \\ k_{1,0} \oplus a_{1,0} & k_{1,1} \oplus a_{1,1} & k_{1,2} \oplus a_{1,2} & k_{1,3} \oplus a_{1,3} \\ k_{2,0} \oplus a_{2,0} & k_{2,1} \oplus a_{2,1} & k_{2,2} \oplus a_{2,2} & k_{2,3} \oplus a_{2,3} \\ k_{3,0} \oplus a_{3,0} & k_{3,1} \oplus a_{3,1} & k_{3,2} \oplus a_{3,2} & k_{3,3} \oplus a_{3,3} \end{pmatrix}$$

- Since this is a $\oplus$ operation, ark^{-1} is the same as ark. We have

$$ark(ark^{-1}(a, K_i), K_i) = ark^{-1}(ark(a, K_i), K_i) = a$$

AES-128 Round Keys

- Let $K = K[0,31]K[32,63]K[64,95]K[96,127]$ be a 4-word encryption key
- AES expands K into a 44-word array $W[0,43]$
- Define a byte transformation function $\mathscr{M}$ as follows:

$$\mathscr{M}(b_7b_6b_5b_4b_3b_2b_1b_0) = \begin{cases} b_6b_5b_4b_3b_2b_1b_00, & \text{if } b_7 = 0, \\ b_6b_5b_4b_3b_2b_1b_00 \oplus 00011011, & \text{if } b_7 = 1 \end{cases}$$

- Next, let j be a non-negative number. Define $m(j)$ as follows:

$$m(j) = \begin{cases} 00000001, & \text{if } j = 0 \\ 00000010, & \text{if } j = 1 \\ \mathscr{M}(m(j-1)), & \text{if } j > 1 \end{cases}$$

- Finally, define a word-substitution function T as follows, which transforms a 32-bit string into a 32-bit string, using parameter j and the AES S-Box: $T(w, j) = [(S(w_2) \oplus m(j - 1)]S(w_3)\ S(w_4)\ S(w_1)$, where $w = w_1w_2w_3w_4$ with each w_i being a byte

Putting Things Together

- Use all of these functions to create round keys of size 4 words (11 round keys are needed for AES-128; i.e. 44 words)

$$W[0] = K[0, 31]$$
$$W[1] = K[32, 63]$$
$$W[2] = K[64, 95]$$
$$W[3] = K[96, 127]$$
$$W[i] = \begin{cases} W[i-4] \oplus T(W[i-1], i/4), \text{ if } i \text{ is divisible by 4} \\ W[i-4] \oplus W[i-1], \text{ otherwise} \end{cases}$$
$$i = 4, \ldots, 43$$

- 11 round keys: For $i = 0, \ldots, 10$:

$$K_i = W[4\text{i}, 4i + 3] = W[4i + 0]\ W[4i + 1]\ W[4i + 2]\ W[4i + 3]$$

AES-128 Encryption/Decryption

- AES-128 encryption:
- Let A_i (i = 0, …, 11) be a sequence of state matrices, where A_0 is the initial state matrix M, and A_i (i = 1, …, 10) represents the input state matrix at round i
- A_{11} is the cipher text block C, obtained as follows:

$$A_1 = ark(A_0, K_0)$$

$$A_{i+1} = ark(mic(shr(sub(A_i))), K_i),\ i = 1,\ldots,9$$

$$A_{11} = arc(shr(sub(A_{10})), K_{10}))$$

- AES-128 decryption:
- Let $C_0 = C = A_{11}$, where C_i is the output state matrix from the previous round

$$C_1 = ark(C_0, K_{10})$$

$$C_{i+1} = mic^{-1}(ark(sub^{-1}(shr^{-1}(C_i)), K_{10-i})),\ i = 1,\ldots,9$$

$$C_{11} = ark(sub^{-1}(\mathrm{shr}^{-1}(C_{10})), K_0)$$

Correctness Proof of Decryption

- We now show that $C_{11} = A_0$
- We first show the following equality using mathematical induction:

  $C_i = shr(sub(A_{11-i})),\ i = 1, \ldots, 10$

  For $i = 1$ we have

$$\begin{aligned} C_1 &= ark(A_{11}, K_{10}) \\ &= A_{11} \oplus K_{10} \\ &= ark(shr(sub(A_{10})), K_{10}) \oplus K_{10} \\ &= (shr(sub(A_{10})) \oplus K_{10}) \oplus K_{10} \\ &= shr(sub(A_{10})) \end{aligned}$$

- Assume that the equality holds for $1 \le i \le 10$. We have

$$\begin{aligned} C_{i+1} &= mic^{-1}(ark(sub^{-1}(shr^{-1}(C_i)), K_{10-i})) \\ &= mic^{-1}(ark(sub^{-1}(shr^{-1}(shr(sub(A_{11-i})))) \oplus K_{10-i})) \\ &= mic^{-1}(A_{11-i} \oplus K_{10-i}) \\ &= mic^{-1}(ark(mic(shr(sub(A_{10-i}))), K_{10-i}) \oplus K_{10-i}) \\ &= mic^{-1}([mic(shr(sub(A_{10-i}))) \oplus K_{10-i}] \oplus K_{10-i}) \\ &= shr(sub(A_{10-i}) \\ &= shr(sub(A_{11-(i+1)})) \end{aligned}$$

- This completes the induction proof

Correctness Proof of Decryption

- Finally, we have

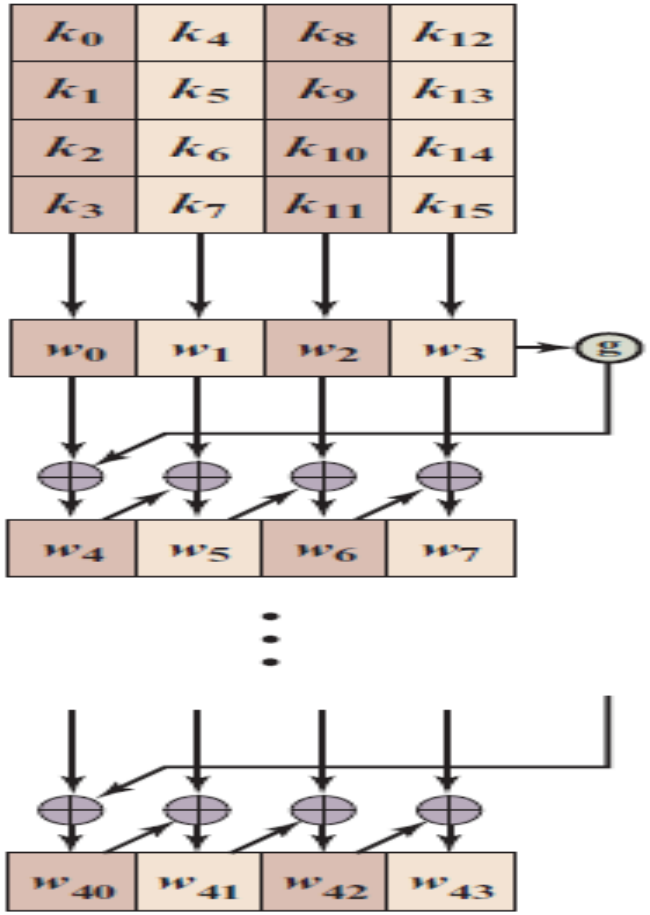
$$
\begin{aligned}
C_{11} &= \text{ark}(\text{sub}^{-1}(\text{shr}^{-1}(C_{10})), K_0) \\
&= \text{sub}^{-1}(\text{shr}^{-1}(\text{shr}(\text{sub}(A_1)))) \oplus K_0 \\
&= A_1 \oplus K_0 \\
&= (A_0 \oplus K_0) \oplus K_0 \\
&= A_0
\end{aligned}
$$

This completes the correctness proof of AES-128 Decryption

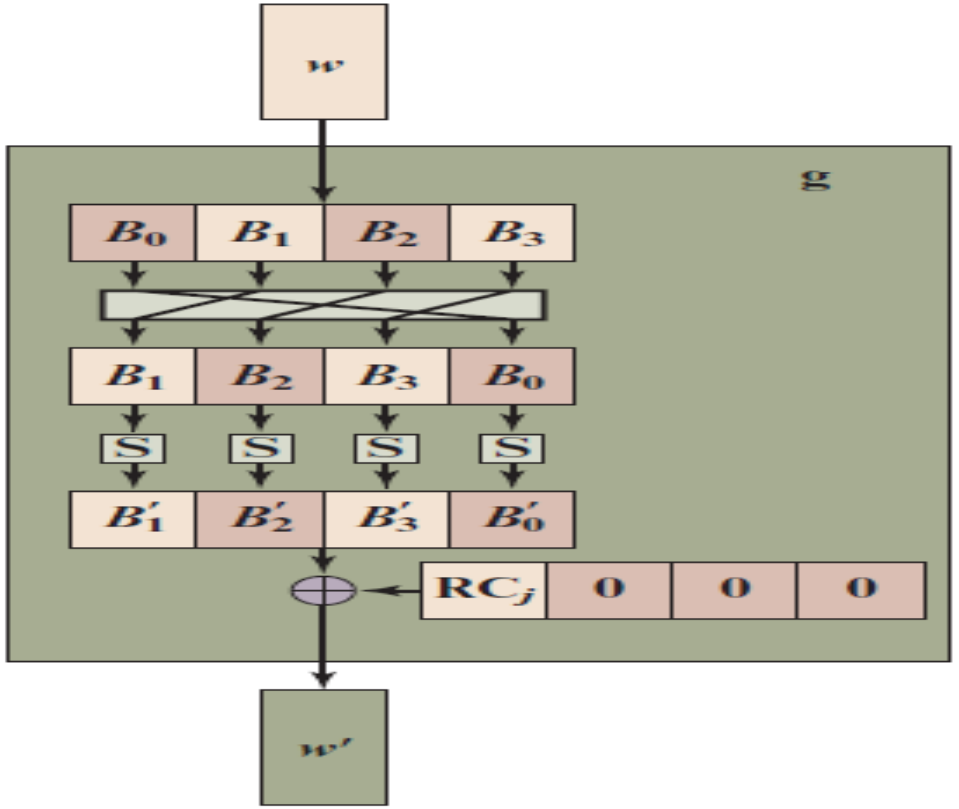
AES Key Expansion

- Takes as input a four-word (16 byte) key and produces a linear array of 44 words (176) bytes
  - This is sufficient to provide a four-word round key for the initial AddRoundKey stage and each of the 10 rounds of the cipher
- Key is copied into the first four words of the expanded key
  - The remainder of the expanded key is filled in four words at a time
- Each added word *w*[i] depends on the immediately preceding word, *w[i – 1]*, and the word four positions back, w[i – 4]
  - In three out of four cases a simple XOR is used
  - For a word whose position in the w array is a multiple of 4, a more complex function is used

AES Key Expansion



(a) Overall algorithm



(b) Function g

Key Expansion Rationale (1 of 2)

- The Rijndael developers designed the expansion key algorithm to be resistant to known cryptanalytic attacks
- Inclusion of a round-dependent round constant eliminates the symmetry between the ways in which round keys are generated in different rounds

Key Expansion Rationale (2 of 2)

- The specific criteria that were used are:
  - Knowledge of a part of the cipher key or round key does not enable calculation of many other round-key bits
  - An invertible transformation
  - Speed on a wide range of processors
  - Usage of round constants to eliminate symmetries
  - Diffusion of cipher key differences into the round keys
  - Enough nonlinearity to prohibit the full determination of round key differences from cipher key differences only
  - Simplicity of description

AES Implementation

- AES decryption cipher is not identical to the encryption cipher
  - The sequence of transformations differs although the form of the key schedules is the same
  - Has the disadvantage that two separate software or firmware modules are needed for applications that require both encryption and decryption
- Two separate changes are needed to bring the decryption structure in line with the encryption structure
- The first two stages of the decryption round need to be interchanged
- The second two stages of the decryption round need to be interchanged

Modes of Operations

- NIST has approved 14 modes

  - ✓ 8 confidentiality modes: ECB, CBC, OFB, CFB, CTR, XTS-AES, FF1, FF3;
  - ✓ 1 authentication mode: CMAC;
  - ✓ 5 combined modes for confidentiality and authentication: CCM, GCM, KW, KWP,TKW

https://csrc.nist.gov/projects/block-cipher-techniques/bcm/current-modes

Topics

- EBC, CBC, CFB, OFB, CTR
- Notes and Remarks on each modes

Modes of Operation

- Block ciphers encrypt fixed size blocks
  - eg. DES encrypts 64-bit blocks, with 56-bit key
  - AES encrypts 128-bit blocks with 128, 192, 256-bit key
- Need way to use in practise, given **usually have arbitrary amount of data to encrypt**
  - Partition message into separate block for ciphering
- A **mode of operation** describes the process of encrypting each of these blocks **under a single key**
- Some modes may use randomized addition input value

Quick History

1981 — Early modes of operation: **ECB, CBC, CFB, OFB**

- DES Modes of operation
  http://www.itl.nist.gov/fipspubs/fip81.htm

2001 — Revised and including **CTR** mode and AES

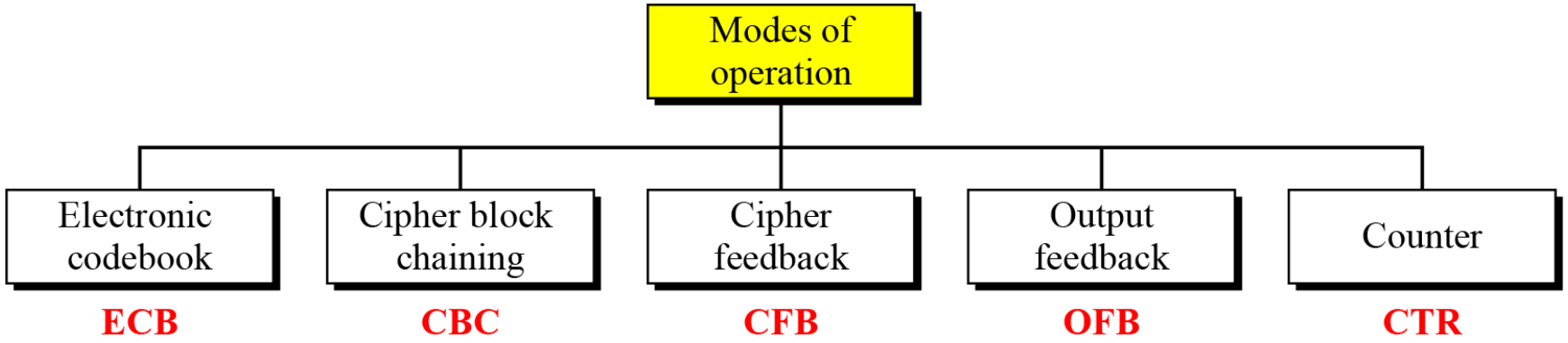
- Recommendation for Block Cipher Modes of Operation
  http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf

2010 — New Mode : **XTS-AES**

- Recommendation for Block Cipher Modes of Operation: The XTS-AES Mode for Confidentiality on Storage Devices
  http://csrc.nist.gov/publications/nistpubs/800-38E/nist-sp-800-38E.pdf

Modes of Operation Taxonomy

- Current well-known modes of operation



Moe 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

$$C_i = E_K(Pi)$$

- Uses: secure transmission of single values

ECB Scheme

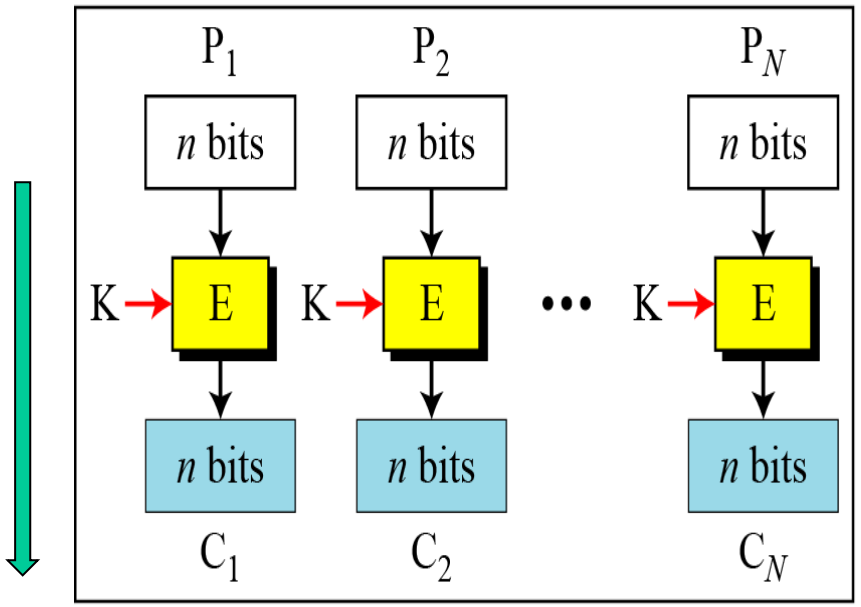
Encryption: $C_i = E_K(P_i)$ Decryption: $P_i = D_K(C_i)$

E: Encryption D: Decryption

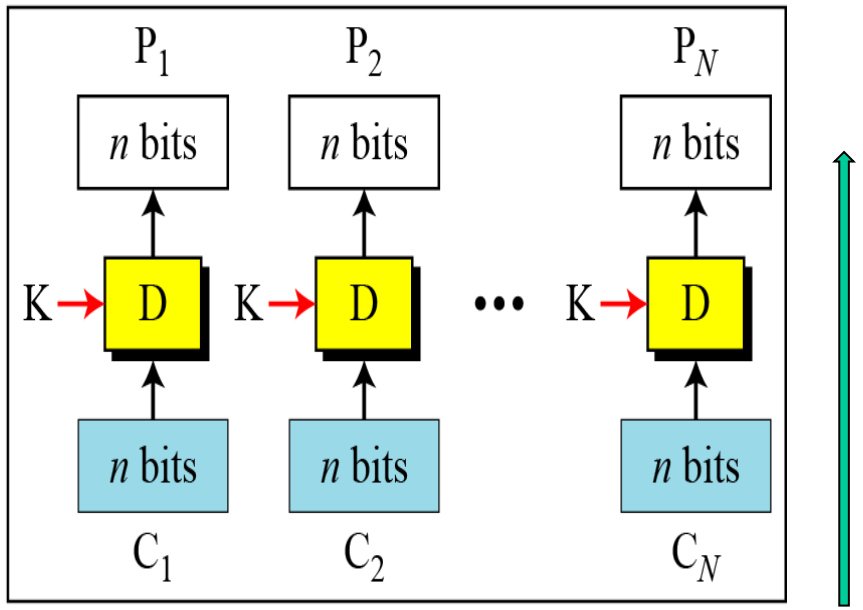
P_i: Plaintext block i C_i: Ciphertext block i

K: Secret key $n = 64 \text{ or } 128$

$P = P_1 || P_2 || \dots | P_n^*$



Encryption



Decryption

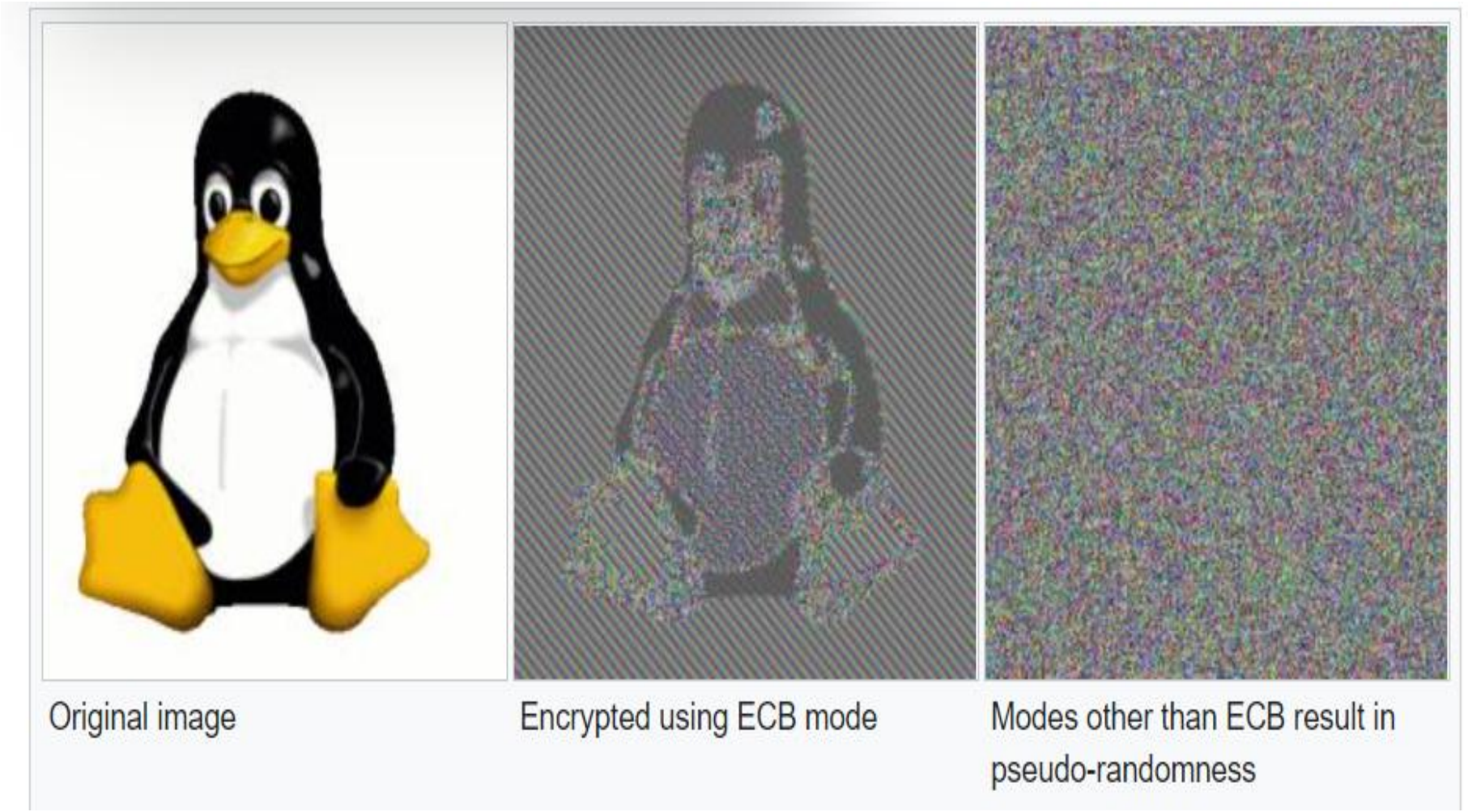
$C = C_1 || C_2 || \dots | C_n$

Remarks on ECB

- Strength: it's simple.
- Weakness:
  - Repetitive data 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)

Remarks on ECB



Original image

Encrypted using ECB mode

Modes other than ECB result in pseudo-randomness

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

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

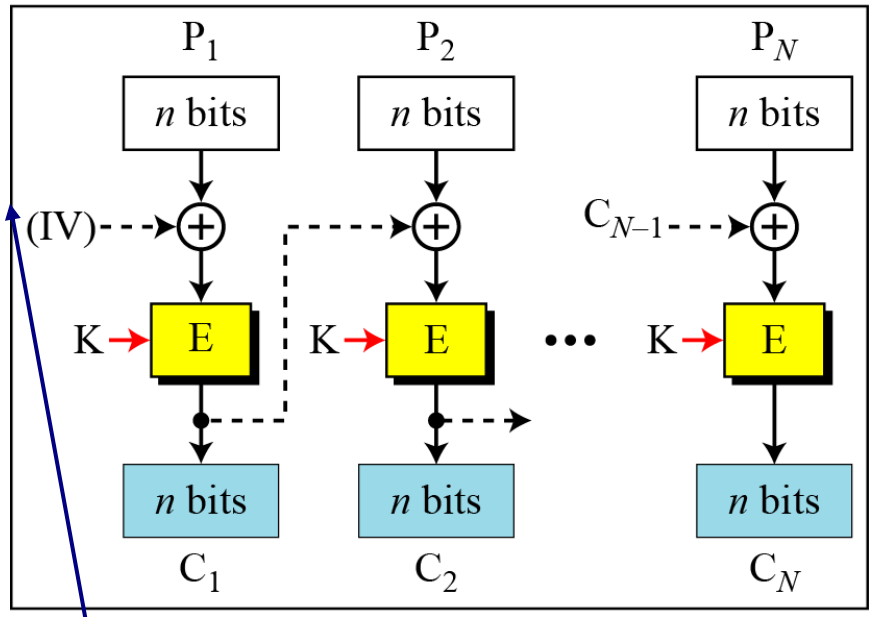
$$C_0 = IV$$

- Uses: bulk data encryption, authentication

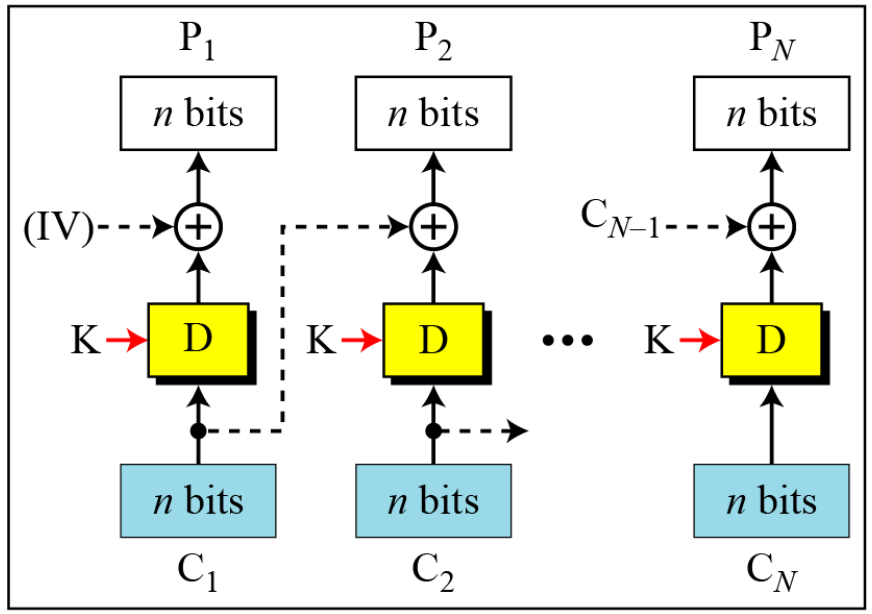
CBC scheme

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

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



Encryption



Decryption

Encryption:

$C_0 = IV$

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

Decryption:

$C_0 = IV$

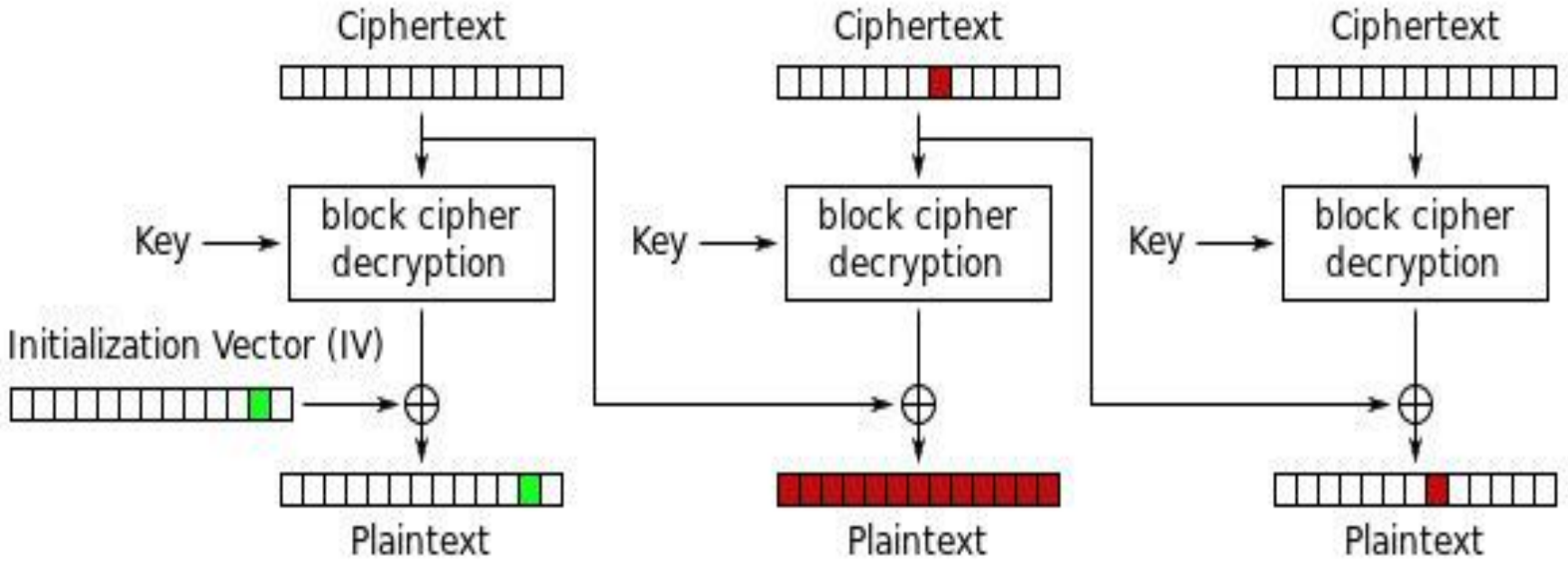
$P_i = D_K (C_i) \oplus C_{i-1}$

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
- Does not guarantee data integrity!

Remarks on CBC

- Does not guarantee data integrity!



Cipher Block Chaining (CBC) mode decryption

https://alicegg.tech/2019/06/23/aes-cbc.html

https://cve.mitre.org/cgi-bin/cvename.cgi?name=2020-8911

Cipher FeedBack (CFB)

- Use Initial Vector to start process
- Encrypt previous ciphertext , then combined with the plaintext block using X-OR to produce the current ciphertext
- Cipher is fed back (hence name) to concatenate with the rest of IV
- Plaintext is treated as a stream of bits
  - Any number of bit (1, 8 or 64 or whatever) to be feed back (denoted CFB-1, CFB-8, CFB-64)
- Relation between plaintext and ciphertext

$$C_i = P_i \oplus \mathbf{SelectLeft}(E_K(\mathbf{ShiftLeft}(C_{i-1})))$$

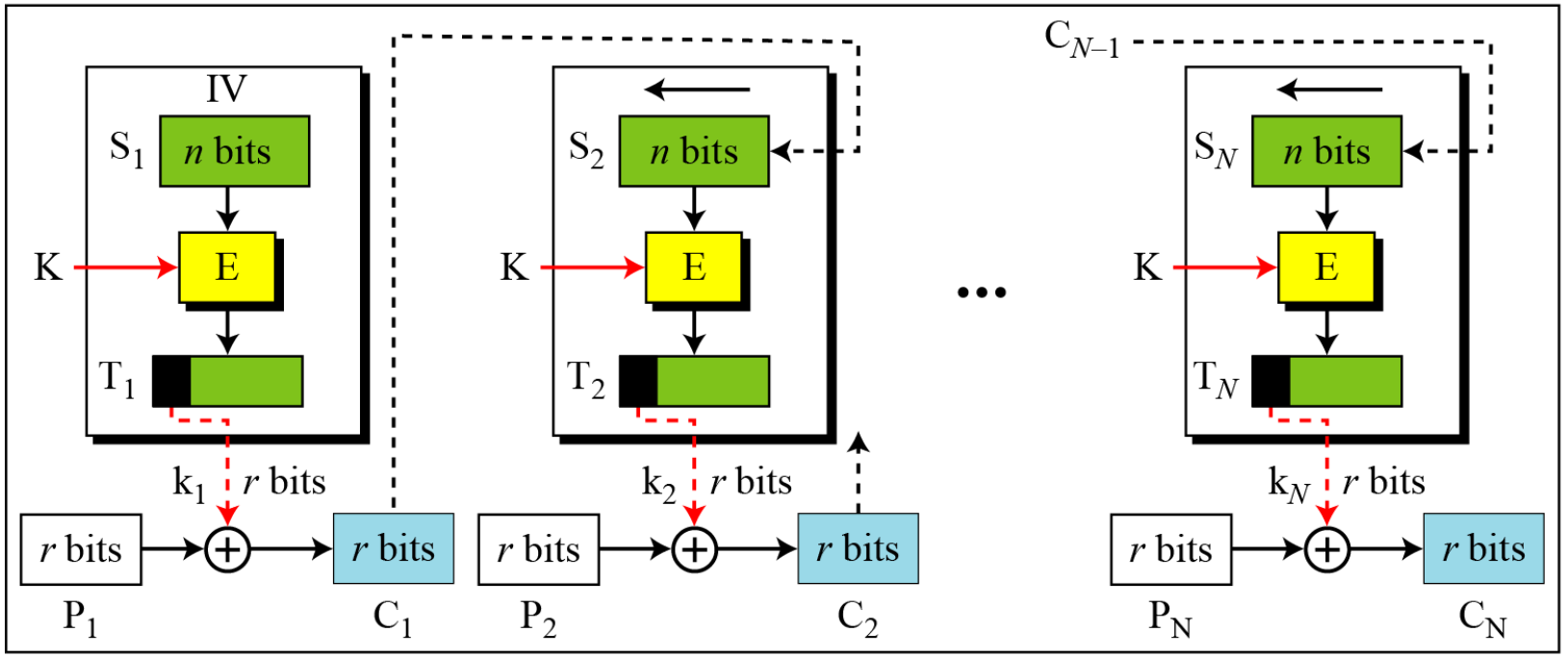
$$C_0 = IV$$

- Uses: stream data encryption, authentication

CFB Scheme

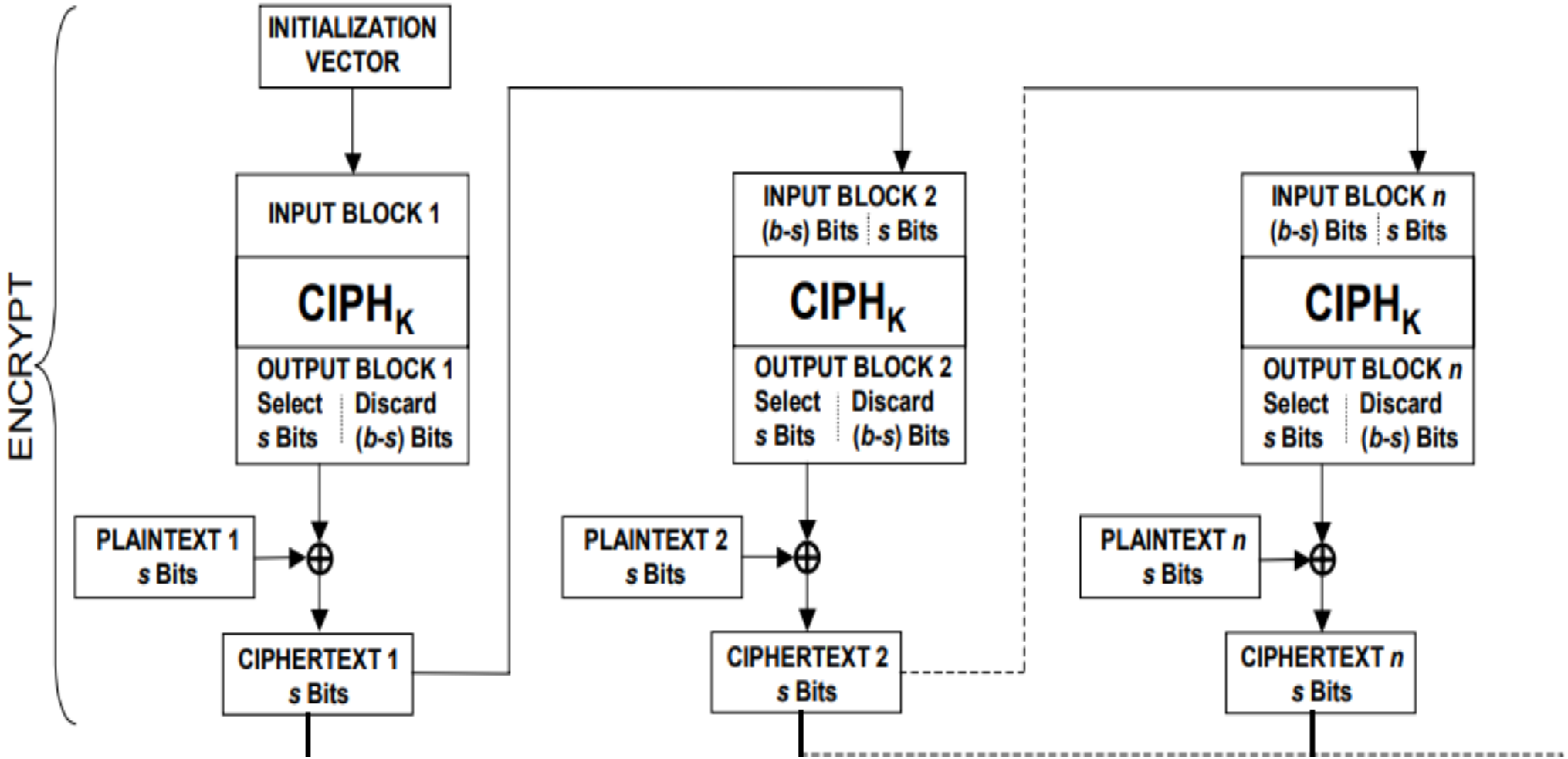
Encryption: $C_i = P_i \oplus \text{SelectLeft}_r \{E_K [\text{ShiftLeft}_r (S_{i-1}) \mid C_{i-1})]\}$
Decryption: $P_i = C_i \oplus \text{SelectLeft}_r \{E_K [\text{ShiftLeft}_r (S_{i-1}) \mid C_{i-1})]\}$

E : Encryption
P_i: Plaintext block i
K: Secret key
D : Decryption
C_i: Ciphertext block i
IV: Initial vector (S_1)
S_i: Shift register
T_i: Temporary register

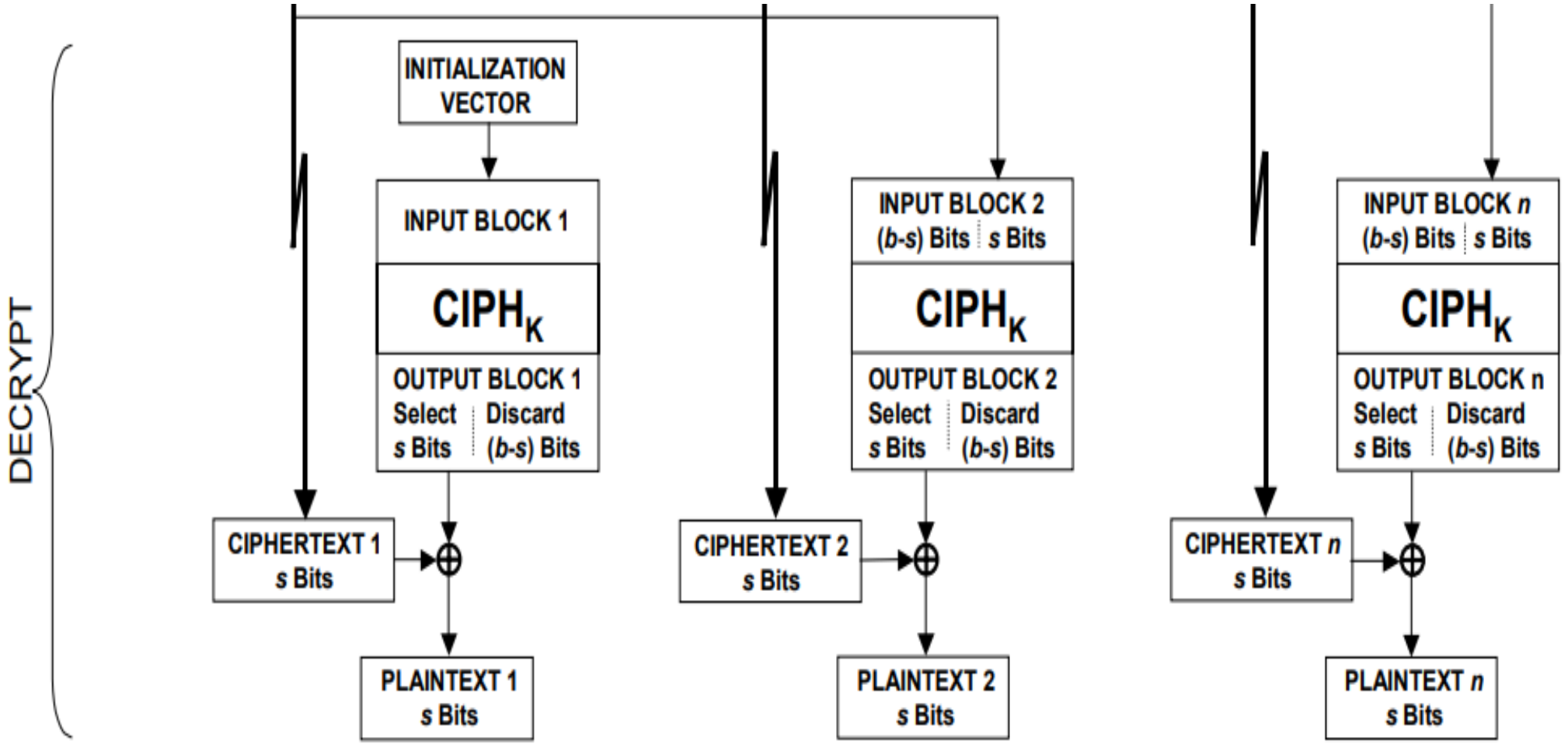


Encryption

CFB Encryption/Decryption

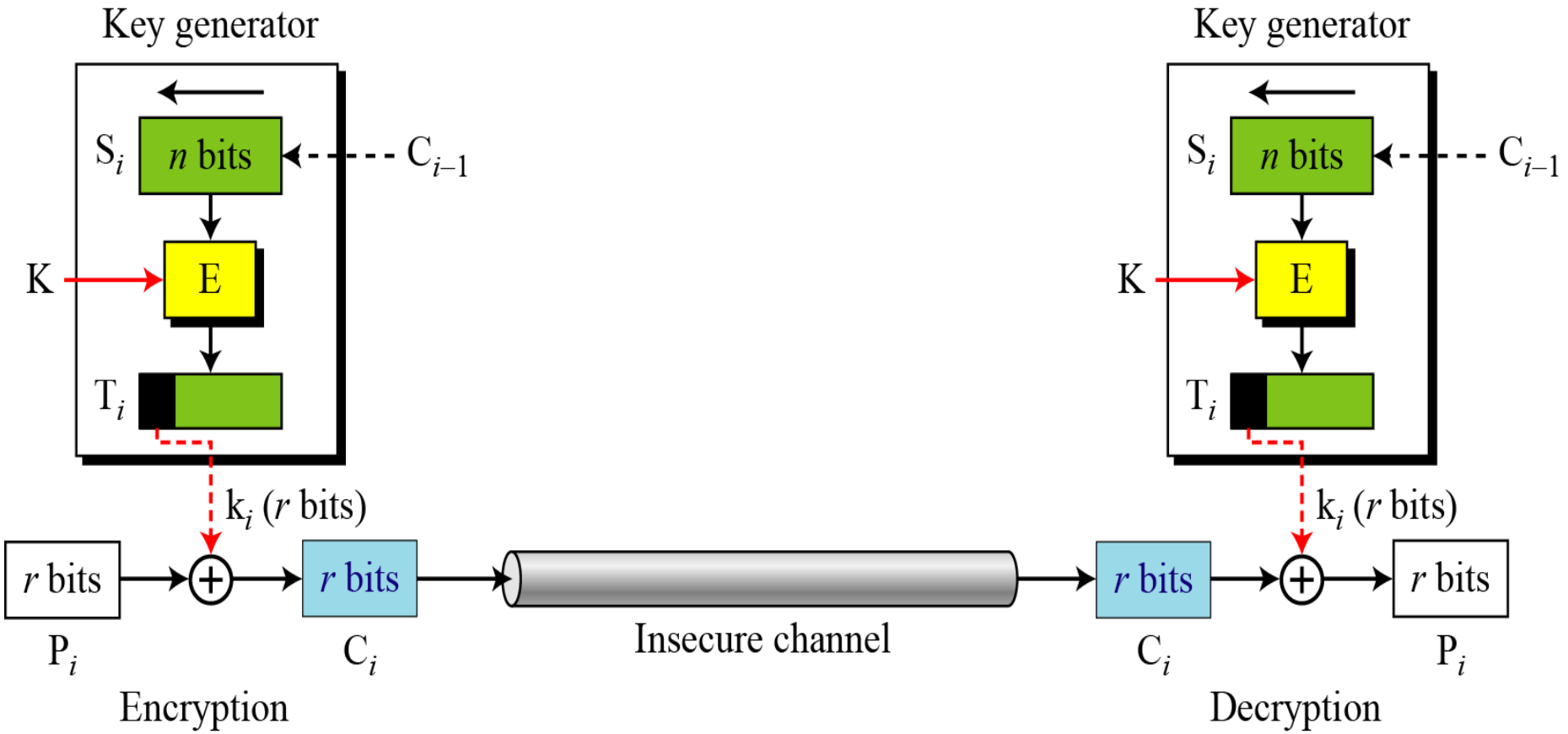


CFB Encryption/Decryption



CFB as a Stream Cipher

- In CFB mode, encipherment and decipherment use the encryption function of the underlying block cipher.



Remark on CFB

- The block cipher is used as a stream cipher.
  - enable to encrypt any number of bits e.g. single bits or single characters (bytes)
  - S=1 : bit stream cipher
  - S=8 : character stream cipher
  - S=64, S=128 (block cipher)
- A ciphertext segment depends on the current and all preceding plaintext segments.
- A corrupted ciphertext segment during transmission will affect the current and next several plaintext segments.

Output FeedBack (OFB)

- Very similar to CFB
- But output of the encryption function output of cipher is fed back (hence name), instead of ciphertext
- Feedback is independent of message
- Relation between plaintext and ciphertext

$$C_i = P_i \oplus O_i$$
$$O_i = E_K (O_{i-1})$$
$$O_0 = IV$$

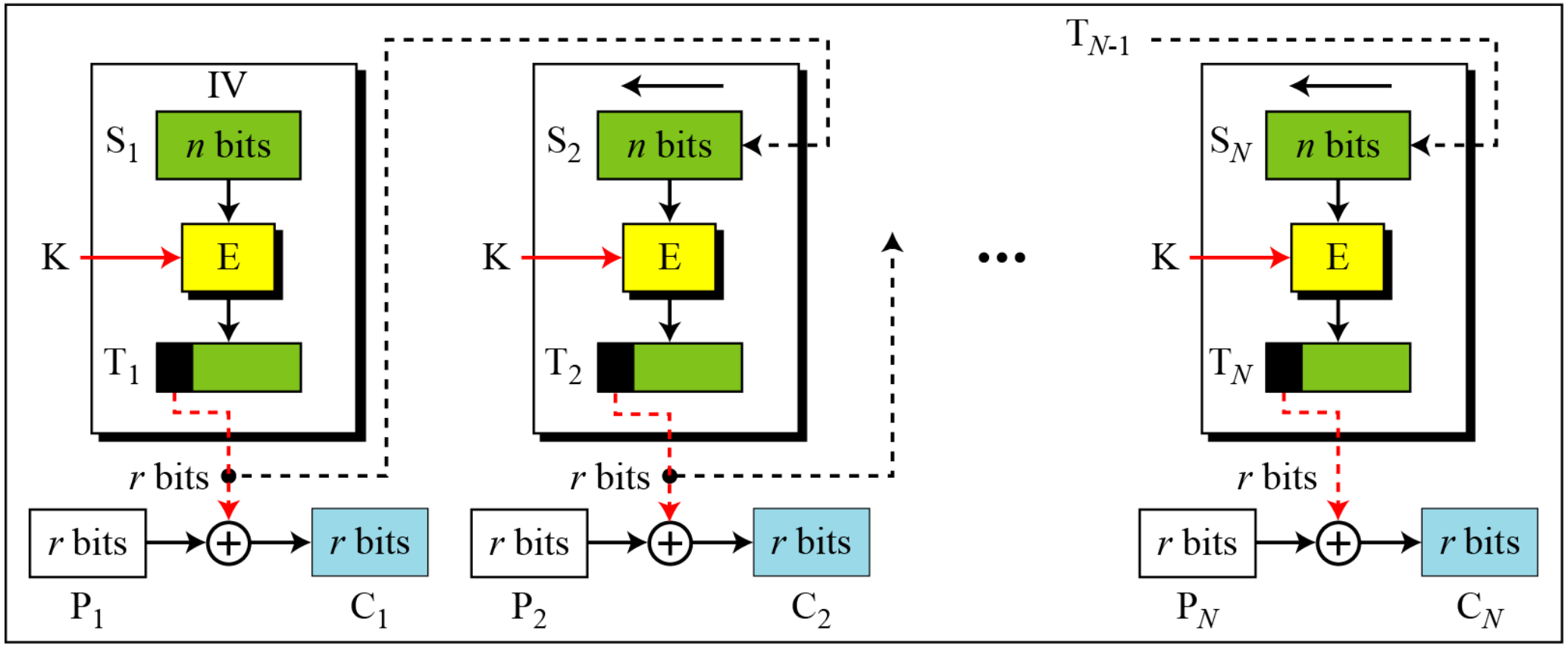
- Uses: stream encryption over noisy channels

OFB Scheme

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

D : Decryption
C_i: Ciphertext block i
IV: Initial vector (S_1)

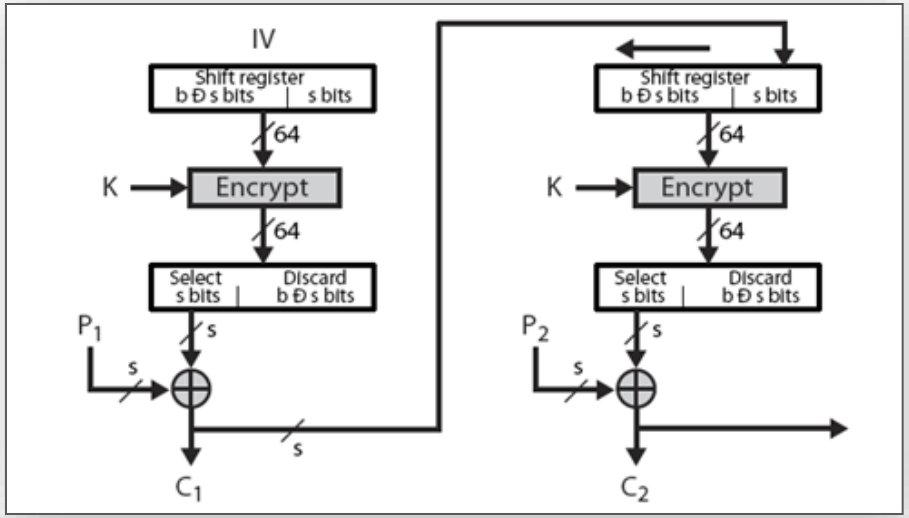
S_i: Shift register
T_i: Temporary register



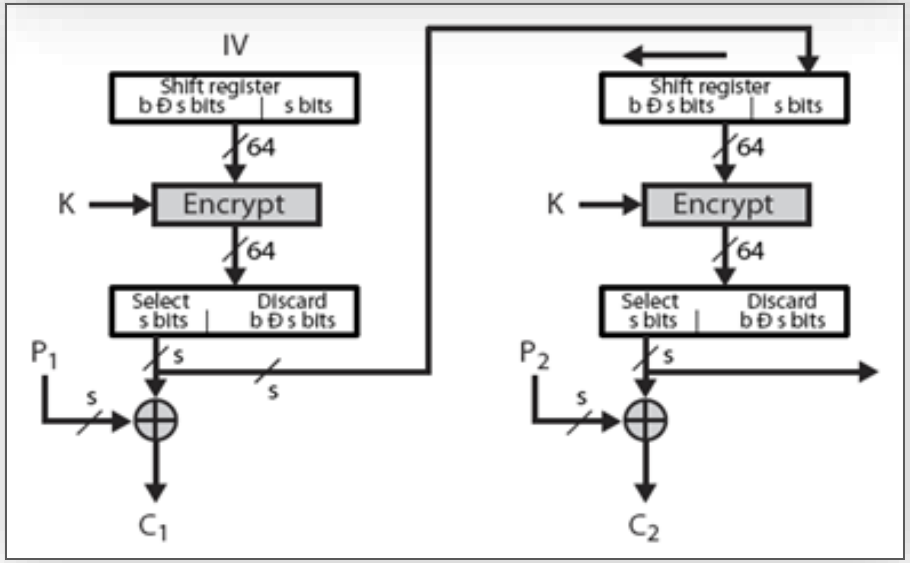
Encryption

CFB V.S. OFB

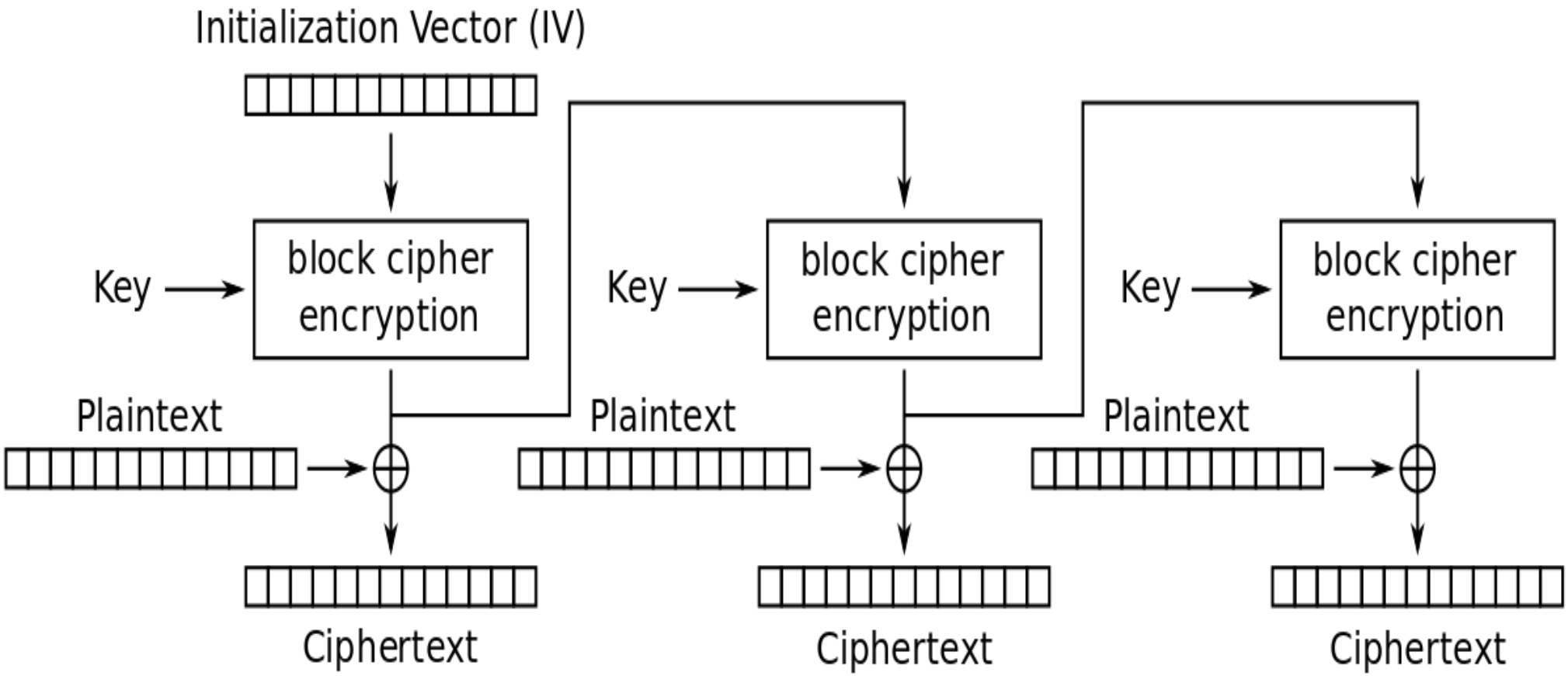
Cipher Feedback



Output Feedback

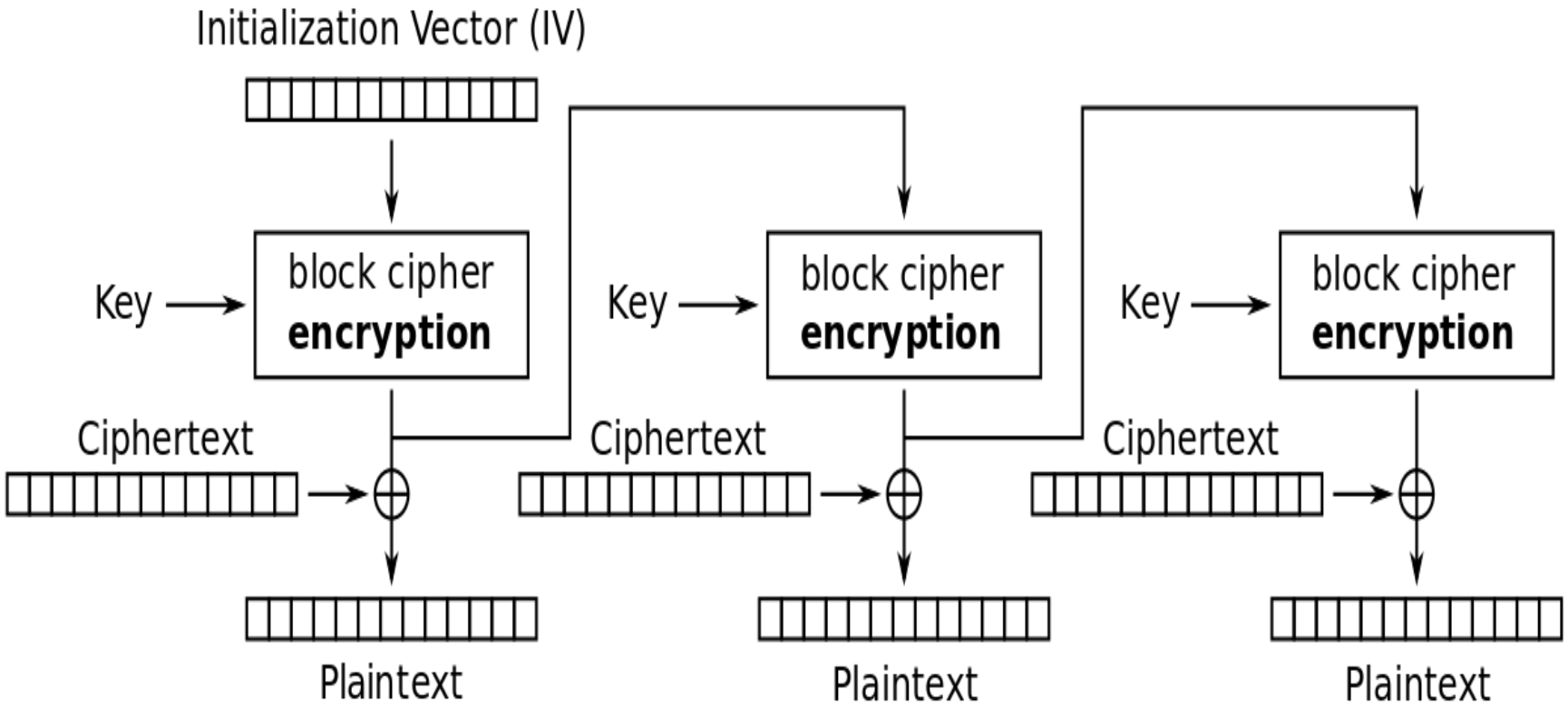


OFB Encryption and Decryption



Output Feedback (OFB) mode encryption

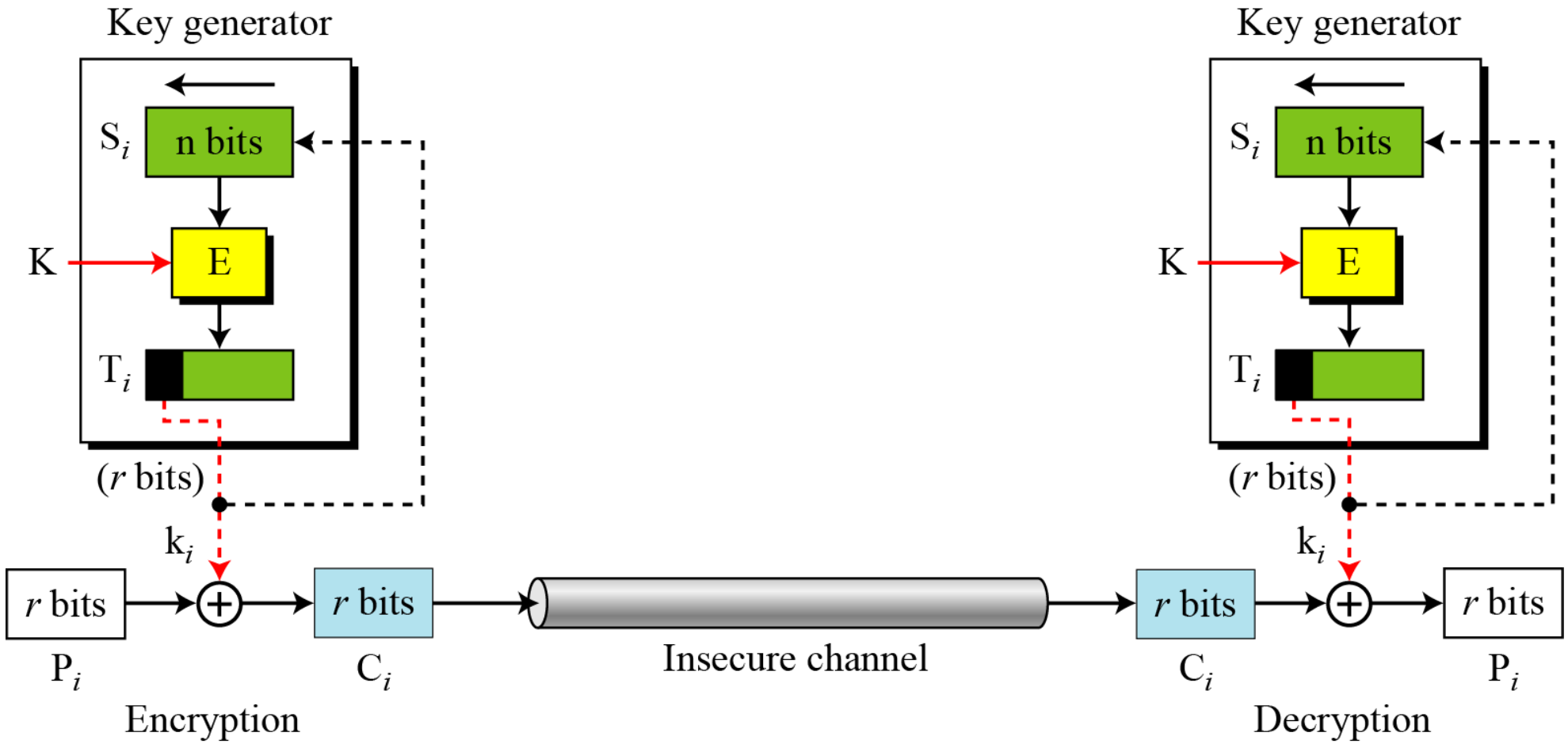
OFB Encryption and Decryption



Output Feedback (OFB) mode decryption

OFB as a Stream Cipher

- In OFB mode, encipherment and decipherment use the encryption function of the underlying block cipher.



Remarks on OFB

- Each bit in the ciphertext is independent of the previous bit or bits. This avoids error propagation
- Pre-compute of forward cipher is possible
- Security issue
  - when j^{th} plaintext is known, the j^{th} output of the forward cipher function will be known
  - Easily cover j^{th} plaintext block of other message with the same IV
- Require that the IV is a nonce

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

$$C_i = P_i \oplus O_i$$
$$O_i = E_K(i)$$

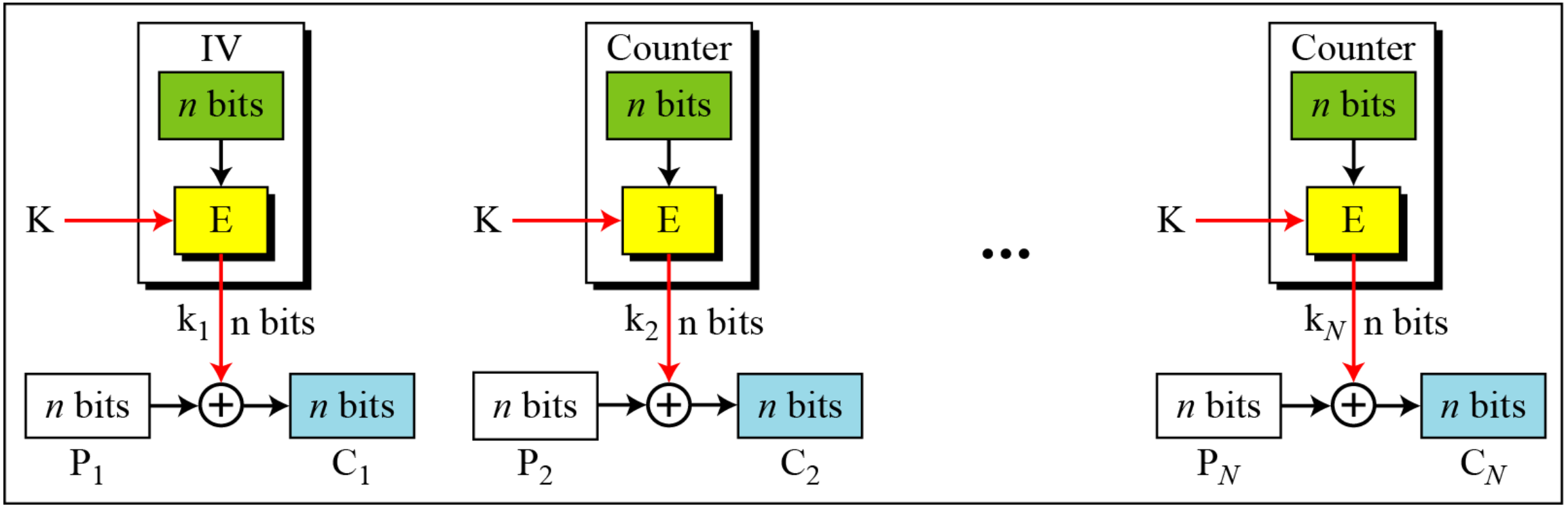
- Uses: high-speed network encryptions

CTR Scheme

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

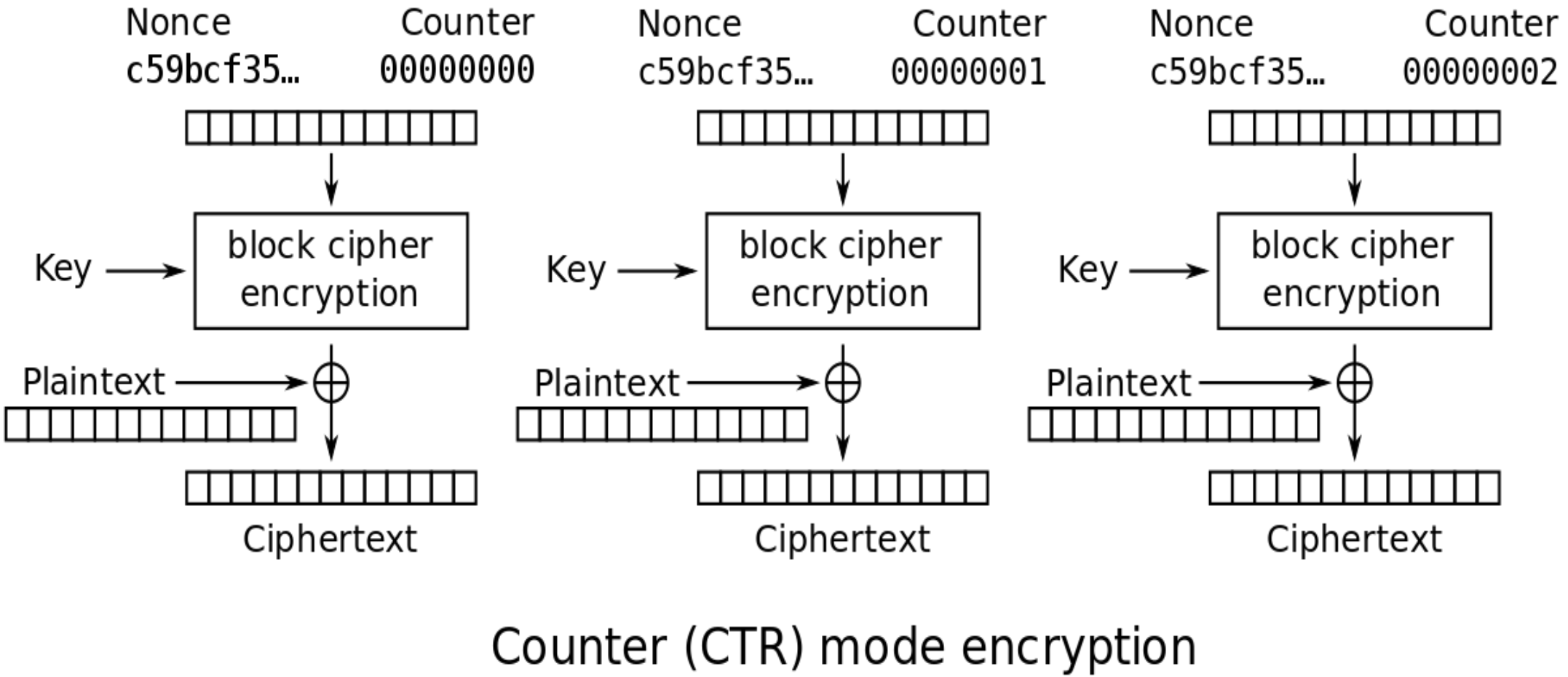
IV: Initialization vector
C_i : Ciphertext block *i*
k_i : Encryption key *i*

The counter is incremented for each block.



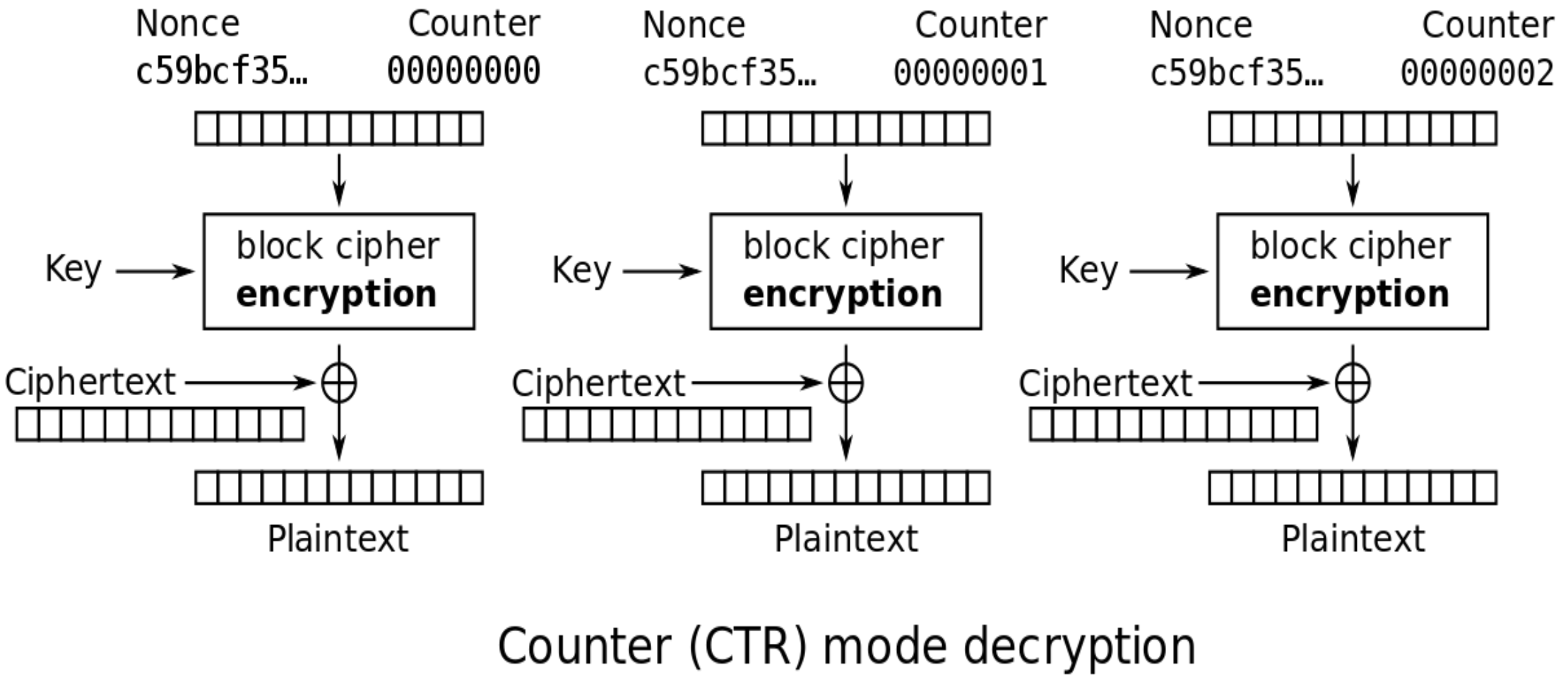
Encryption

CTR Encryption and Decryption



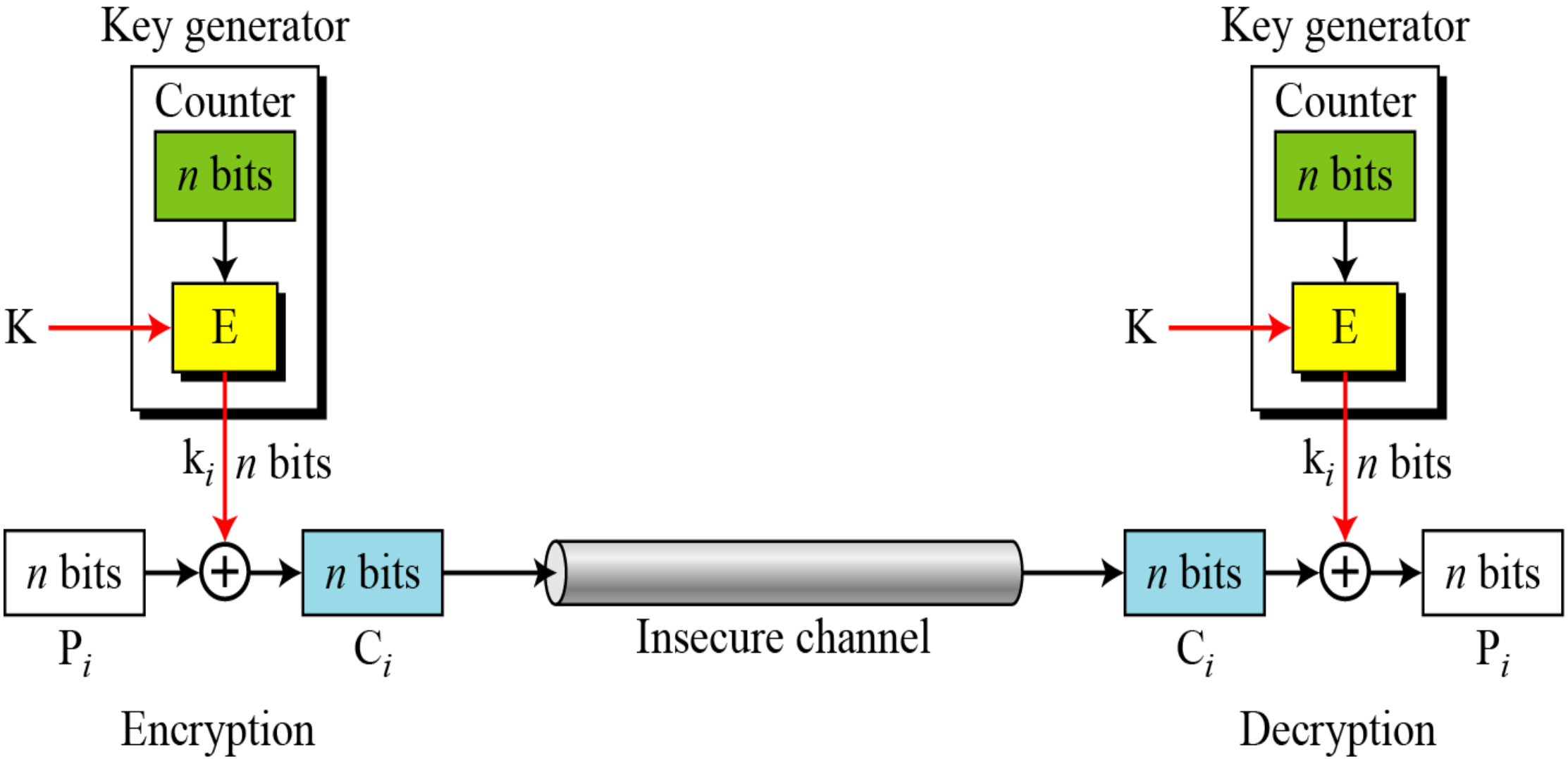
Counter (CTR) mode encryption

CTR Encryption and Decryption



Counter (CTR) mode decryption

OFB as a Stream Cipher



Remark on CTR

- Strengthes:
  - Needs only the encryption algorithm
  - Random access to encrypted data blocks
    - blocks can be processed (encrypted or decrypted) in parallel
  - Simple; fast encryption/decryption

- Counter must be
  - Must be unknown and unpredictable
  - pseudo-randomness in the key stream is a goal

Remark on each mode

- Basically two types:
  - block cipher
  - stream cipher
- CBC is an excellent block cipher
- CFB, OFB, and CTR are stream ciphers
- CTR is faster because simpler and it allows parallel processing

Modes and IV

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

Operation Mode	*Description*	*Type of Result*	*Data Unit Size*
ECB	Each n-bit block is encrypted independently with the same cipher key.	Block cipher	n
CBC	Same as ECB, but each block is first exclusive-ored with the previous ciphertext.	Block cipher	n
CFB	Each r-bit block is exclusive-ored with an r-bit key, which is part of previous cipher text	Stream cipher	$r \leq n$
OFB	Same as CFB, but the shift register is updated by the previous r-bit key.	Stream cipher	$r \leq n$
CTR	Same as OFB, but a counter is used instead of a shift register.	Stream cipher	n

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
CFB	s bits are processed at a time and used similar to CBC	Primary stream cipher. Used for authentication

Comparison of Modes

Mode	Description	Application
OFB	Similar to CFB except that the output is fed back	Stream cipher well suited for transmission over noisy channels
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

Final Notes

- ECB, CBC, OFB, CFB, CTR, and XTS modes only provide confidentiality
- To ensure an encrypted message is not accidentally modified or maliciously tampered requires a separate Message Authentication Code (MAC)
- Several MAC schemes
  - HMAC, CMAC and GMAC
- But.. compositing a confidentiality mode with an authenticity mode could be difficult and error prone
- New modes combined confidentiality and data integrity into a single cryptographic primitive
  - CCM, GCM, CWC, EAX, IAPM and OCB