Cryptographic Algorithms for Data Security: AES & Blowfish Overview

Presented By :
Anurag Parashar
(CSS)

Introduction to Symmetric Key Cryptography

A cryptographic technique where the **same key** is used for both **encryption and decryption.**

Also known as **Secret Key Cryptography.**

Commonly used in applications requiring **fast and efficient encryption** (e.g., AES, Blowfish).

How It Works

- A **single key** is shared between sender and receiver.

- Both parties use this key to **encrypt and decrypt** data.

- Example flow:

Plaintext + Key → 🔒 Encryption → Ciphertext
Ciphertext + Same Key → 🔓 Decryption → Plaintext

Blowfish Algorithm

Inventor: Bruce Schneier (1993)
Purpose: Alternative to DES — faster, stronger, and not patented
Type: Symmetric Block Cipher

blockSize: 64-bits

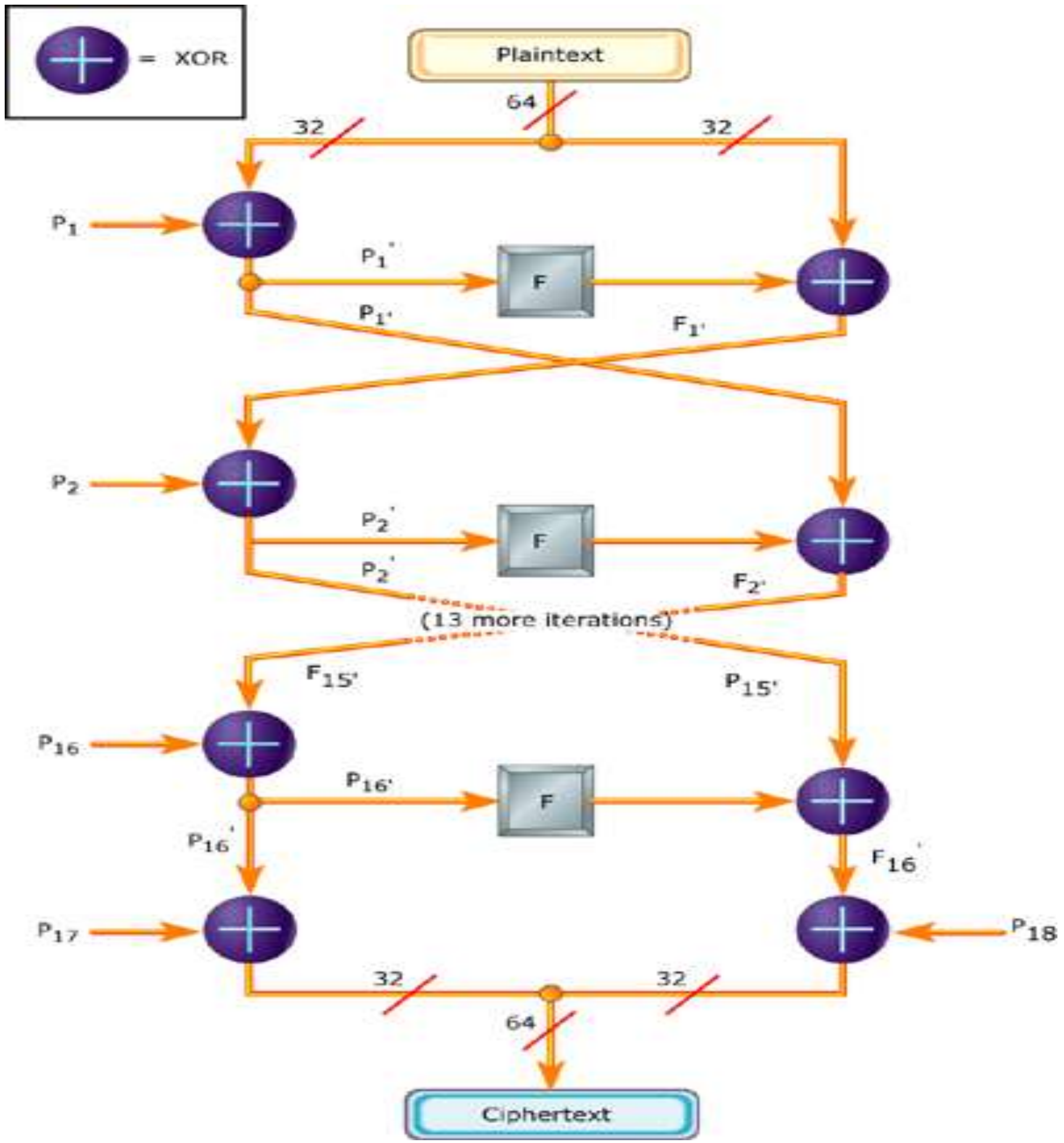
keySize: 32-bits to 448-bits variable size

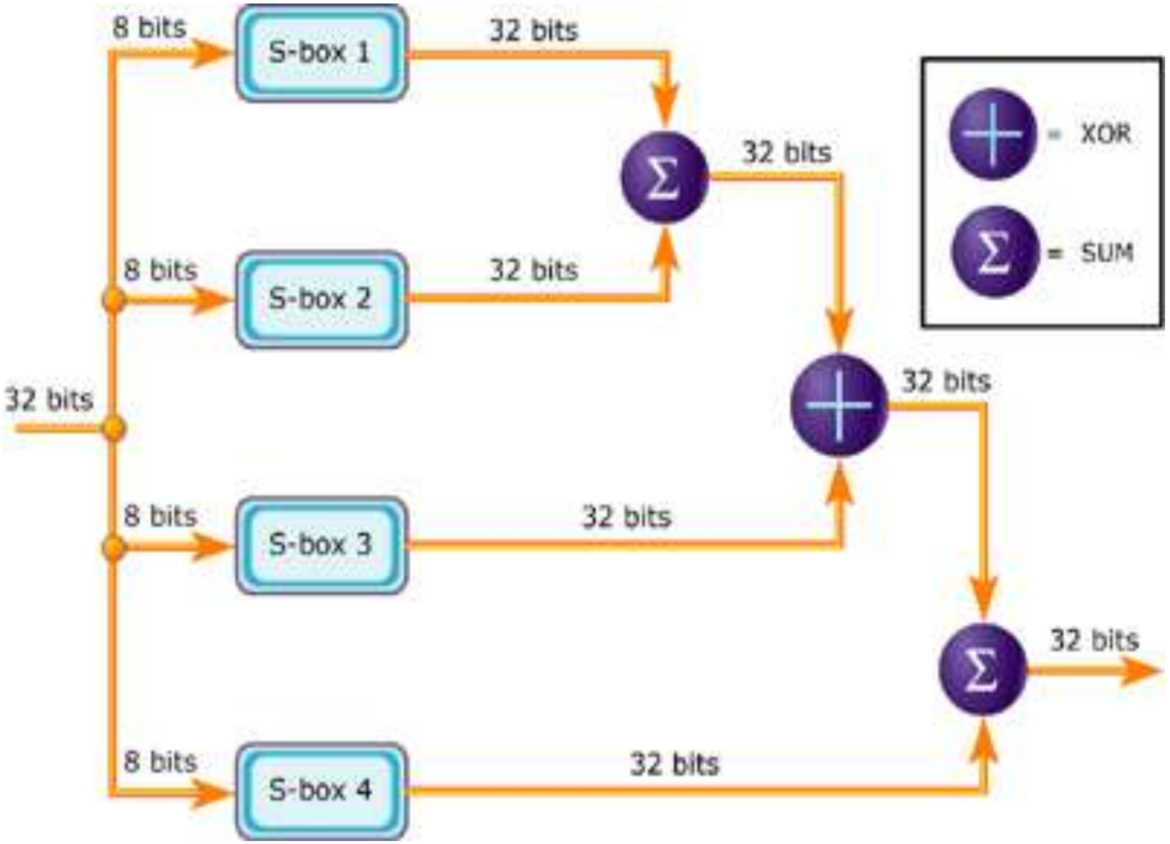
Number of subkeys: 18 [P-array]

Number of rounds: 16

number of substitution boxes: 4 [each having 512 entries of 32 bits each]

Schematic of Blowfish Algorithm





Step 1: Generation of subkeys

- **18 subkeys (P[0] to P[17])** are required.
- Stored in a **P-array**, each of **32 bits.**
- These subkeys are used in **both encryption and decryption** (same for both directions).

32-bit hexadecimal representation of initial values of sub-keys

P[0]	243f6a88	P[9]	38d01377
P[1]	85a308d3	P[10]	be5466cf
P[2]	13198a2e	P[11]	34e90c6c
P[3]	03707344	P[12]	c0ac29b7
P[4]	a4093822	P[13]	c97c50dd
P[5]	299f31d0	P[14]	3f84d5b5
P[6]	082efa98	P[15]	b5470917
P[7]	ec4e6c89	P[16]	9216d5d9
P[8]	452821e6	P[17]	8979fb1b

Now each of the subkey is changed with respect to the input key as:

P[0] = P[0] xor 1st 32-bits of input key
P[1] = P[1] xor 2nd 32-bits of input key.
.
.
.
P[i] = P[i] xor (i+1)th 32-bits of input key (roll over to 1st 32-bits depending on the key length).
.
.
.
P[17] = P[17] xor 18th 32-bits of input key (roll over to 1st 32-bits depending on key length)

The resultant P-array holds 18 subkeys that is used during the entire encryption process

⚙️ Step 2: Initialize Substitution Boxes (S-boxes)

- **S-boxes (Substitution boxes)** are used in each encryption round to perform **complex substitutions** on the data.
- Blowfish uses **4 S-boxes**:

Blowfish uses **4 S-boxes**:

S[0], S[1], S[2], S[3] → Each S-box contains **256 entries** (0 to 255), and each entry is **32 bits** wide.

Usage in Encryption & Decryption

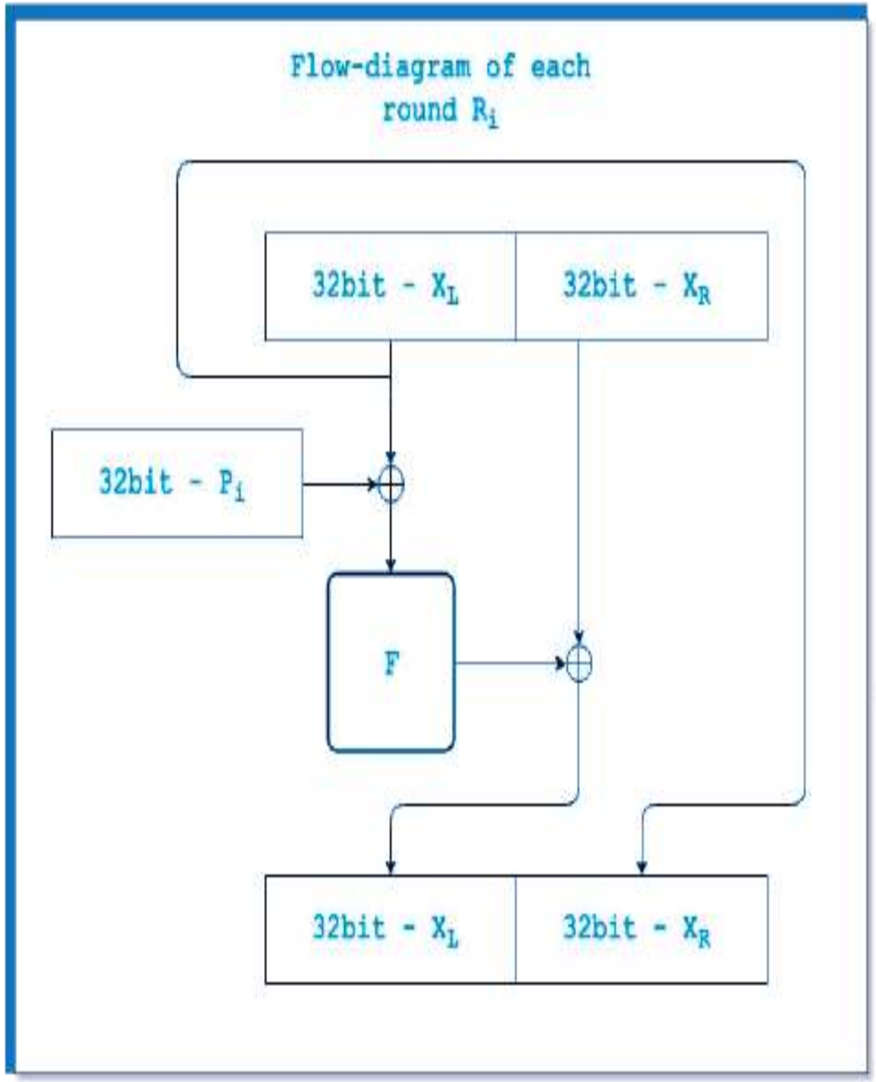
- These S-boxes are used in the **F-function** of Blowfish during **each of the 16 rounds.**
- The same S-boxes are used for both **encryption and decryption**, ensuring reversibility.

Step 3: Encryption

The encryption function consists of two parts:

a. Rounds: The encryption consists of 16 rounds with each round(Ri) taking inputs the plaintext(P.T.) from previous round and corresponding subkey(Pi). The description of each round is as follows

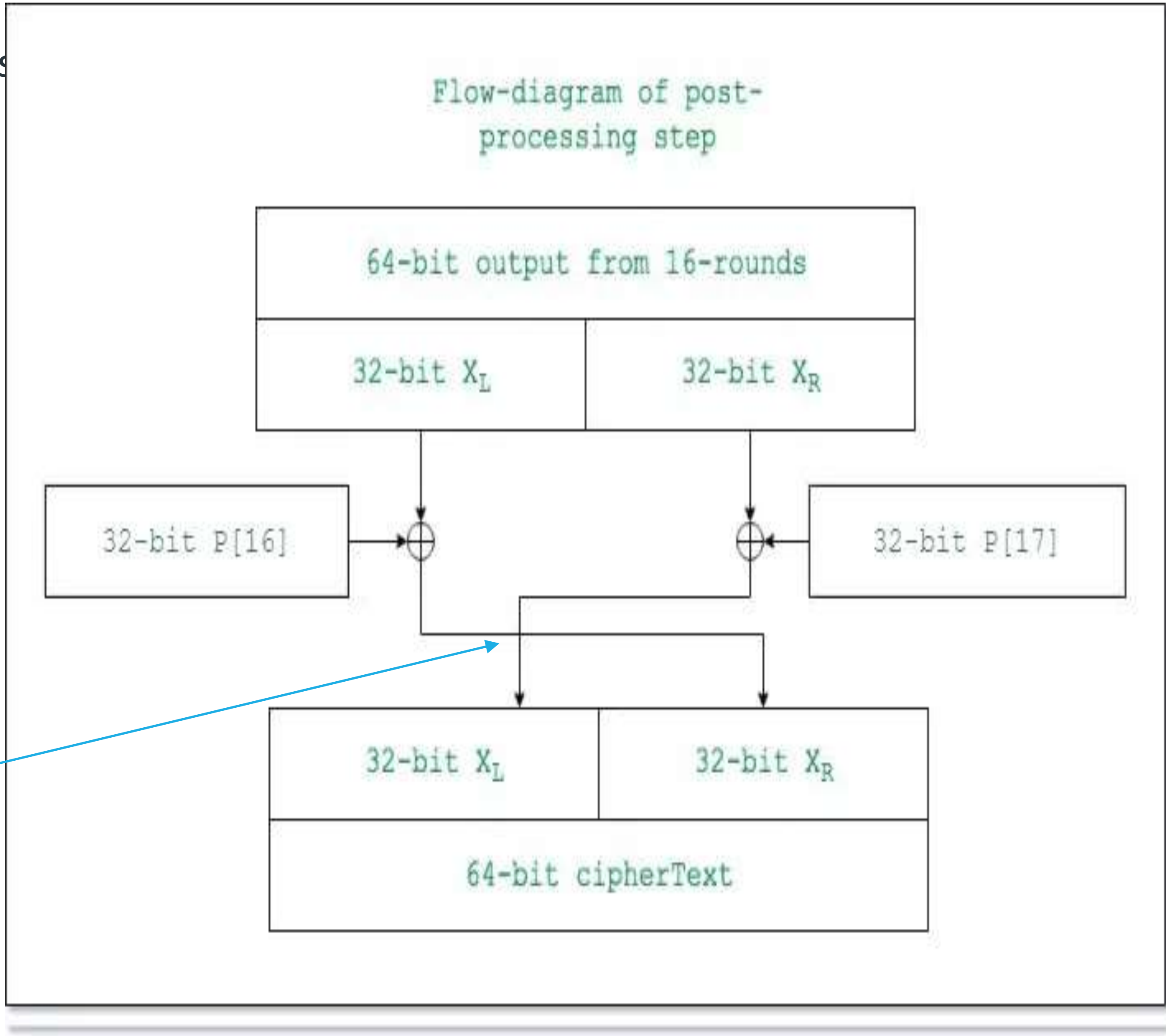
Here, the function "add" is addition modulo 2^32.



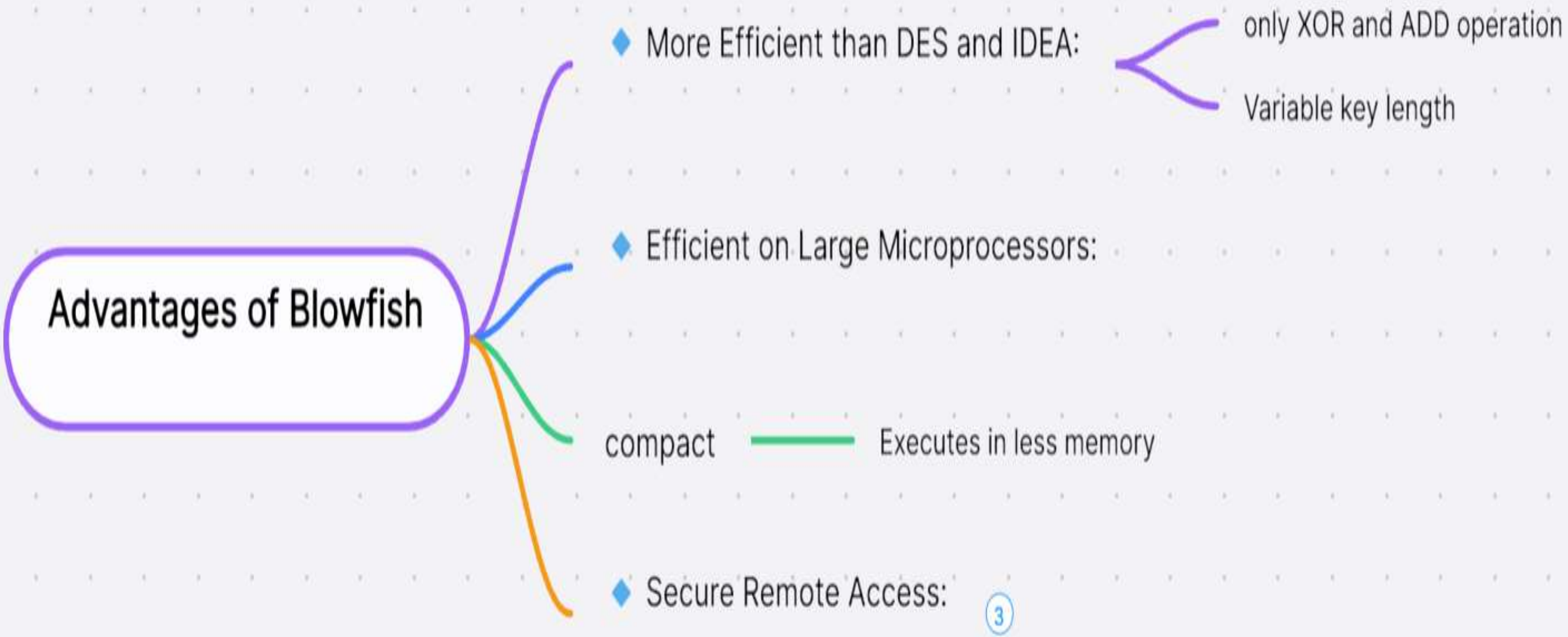
Decryption of Blowfish

In Blowfish, decryption is carried out by reversing the encryption process. Therefore, everything reverses until the ciphertext is converted back into plaintext.

b. Post-processing: The output after the 16 rounds
processed as follows:



Swap L and R


Advantages of Blowfish
More Efficient than DES and IDEA:
only XOR and ADD operation
Variable key length
Efficient on Large Microprocessors:
compact
Executes in less memory
Secure Remote Access:
3

Advanced Encryption Standard (AES)

Overview

•Developed by **NIST in 2001.**

•**Block cipher** used to **secure data** by transforming it into an unreadable format.

•**Widely adopted** due to its strength and reliability

Key Features

•**Key Sizes**: 128, 192, or 256 bits.

•**Block Size**: 128 bits (input and output).

•**Type**: Substitution-Permutation Network (SPN).

•**Rounds**:

- •10 rounds for 128-bit key
- •12 rounds for 192-bit key
- •14 rounds for 256-bit key

Why AES?

•**Much stronger** than DES & 3DES.

•**Efficient** for both hardware and software.

•**Global standard** for:

- Internet security
- File encryption
- Securing sensitive data

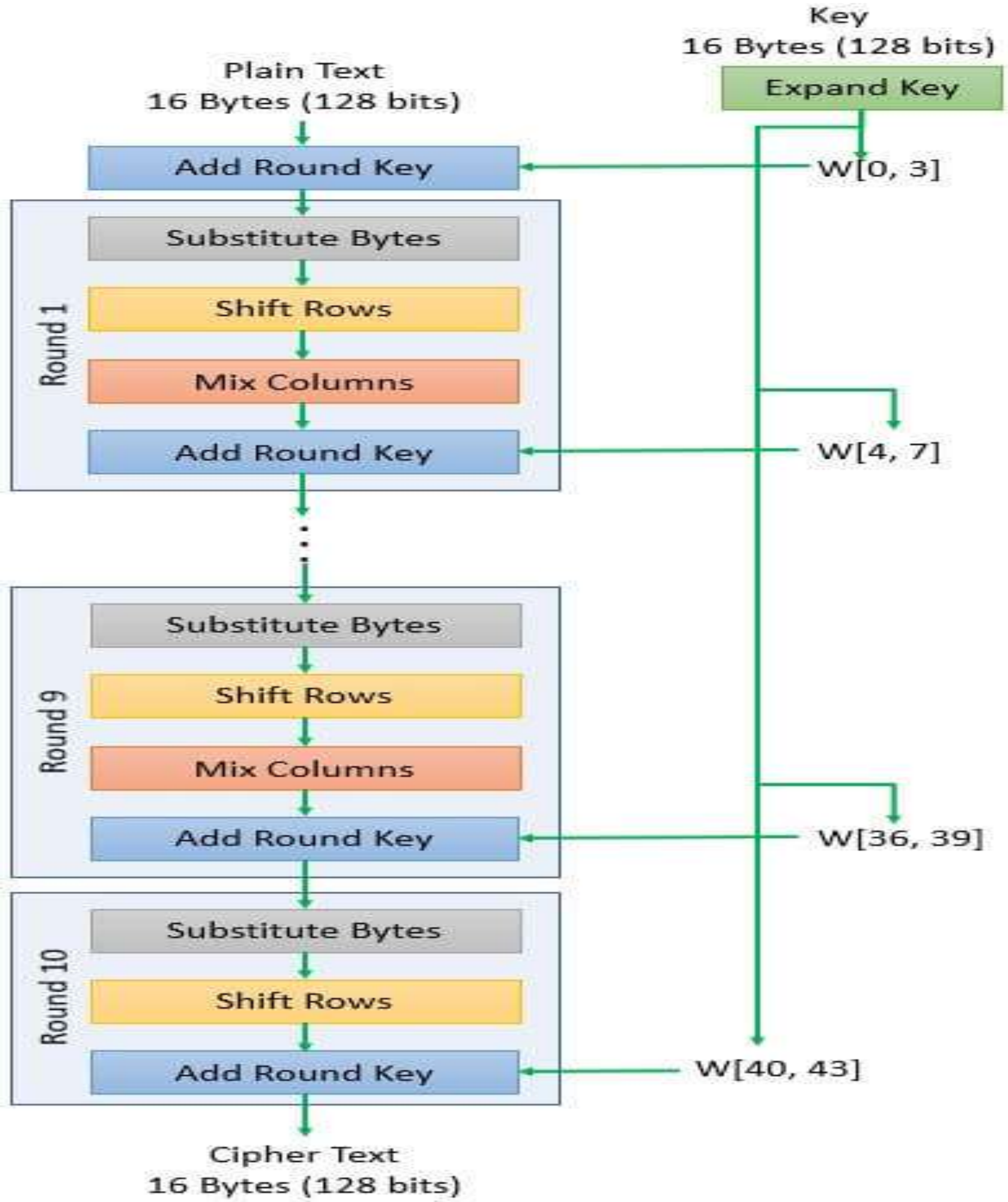
Schematic of AES structure

The AES algorithm can be broken into three phases: the initial round, the main rounds, and the final round.

Initial Round
- **AddRoundKey**

Main Rounds (1 to Nr−1)
- **SubBytes**
- **ShiftRows**
- **MixColumns**
- **AddRoundKey**

Final Round (Nr)
- **SubBytes**
- **ShiftRows**
- **AddRoundKey**



AES Key Schedule

- **Key Schedule** generates a set of **round keys** from the initial secret key.
- The number of **round keys = Nr + 1**:
- 11 keys for 128-bit
- 13 keys for 192-bit
- 15 keys for 256-bit

11 subkeys? But there are only 10 rounds ! That's because first key K0 *is* XOR'd with the plaintext *before* the first round.

- Round keys are used in the **AddRoundKey** step of each round.

AES Key Schedule for 128 bits

$K0=[w0,w1,w2,w3]$

Then each new subkey depends on the previous subkey. To compute

K1=[w4,w5,w6,w7]

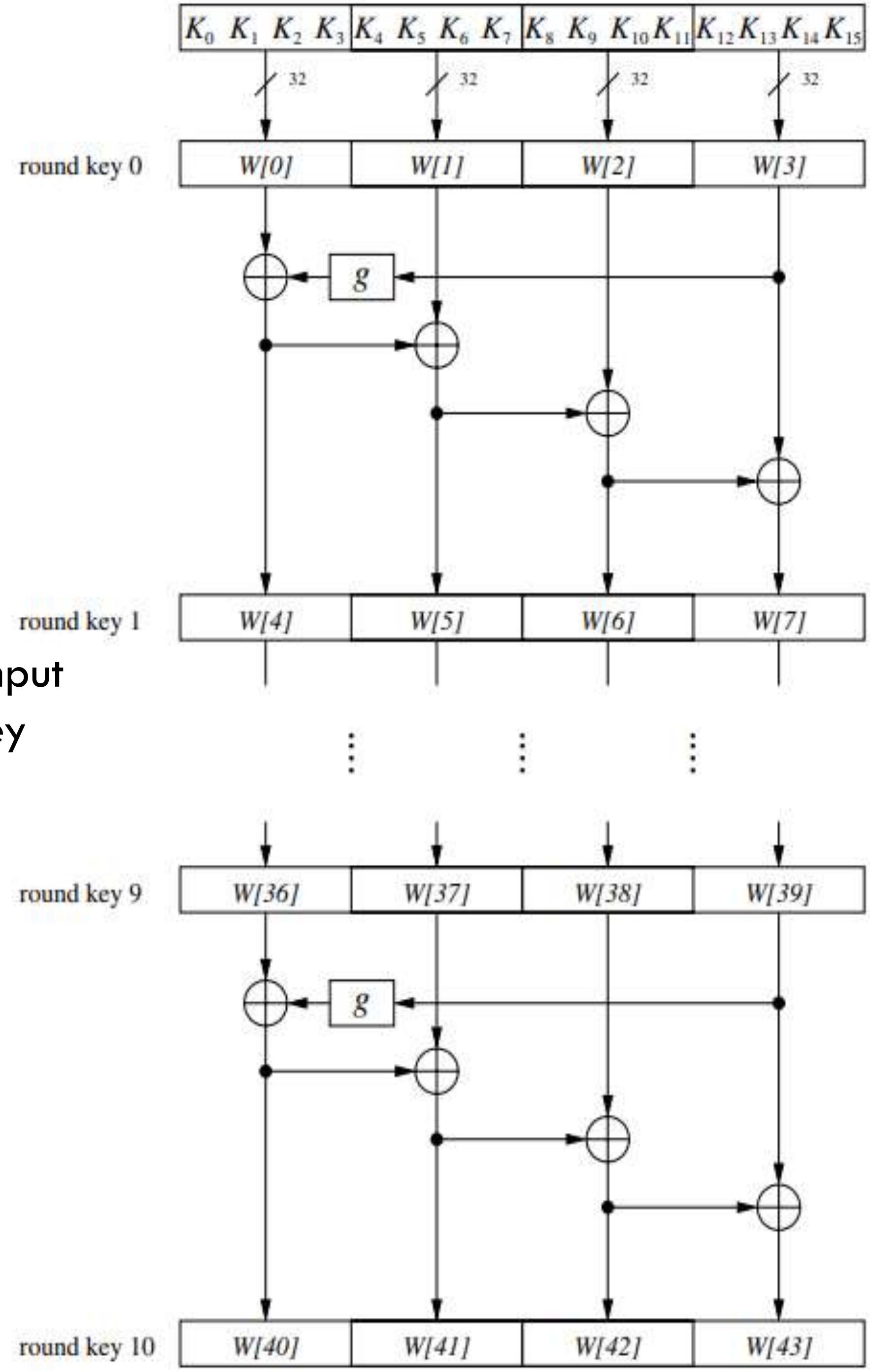
the algorithm the following:

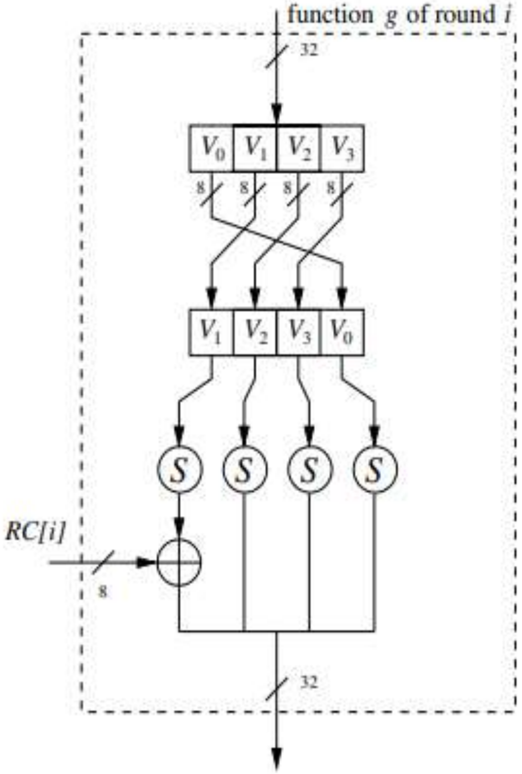
$$W[4i] = W[4(i-1)] + g(W[4i-1]).$$

Here g() is a nonlinear function with a four-byte input and output. The remaining three words of a subkey are computed recursively as:

$$W[4i+j] = W[4i+j-1] + W[4(i-1)+j],$$

where i = 1,....,10 and j = 1,2,3. The function g() rotates its four input bytes





How g() function work

It consists of 3 steps:

RotWord:

Takes a 4-byte word and rotates it left by 1 byte.

Example: Input = [a0, a1, a2, a3]

Output = [a1, a2, a3, a0]

SubWord

Applies the AES S-box to each byte of the word.

Introduces non-linearity and confusion.

Rcon

XORs the result with a round constant (Rcon).

Ensures each round key is uniquely dependent on the round number.

Values of Rcon (for AES-128):

$RC[1] = x^0 = (00000001)_2$

$RC[2] = x^1 = (00000010)_2$

$RC[3] = x^2 = (00000100)_2$

$RC[4] = x^3 = (00001000)_2$

$RC[5] = x^4 = (00010000)_2$

$RC[6] = x^5 = (00100000)_2$

$RC[7] = x^6 = (01000000)_2$

$RC[8] = x^7 = (10000000)_2$

$RC[9] = x^8 = (00011011)_2$

$RC[10] = x^9 = (00110110)_2$

AES Example - Input (128 bit key and message)

Key in English: Thats my Kung Fu (16 ASCII characters, 1 byte each)

Translation into Hex:

T	h	a	t	s		m	y		K	u	n	g		F	u
54	68	61	74	73	20	6D	79	20	4B	75	6E	67	20	46	75

Key in Hex (128 bits): 54 68 61 74 73 20 6D 79 20 4B 75 6E 67 20 46 75

Plaintext in English: Two One Nine Two (16 ASCII characters, 1 byte each)

Translation into Hex:

T	w	o		O	n	e		N	i	n	e		T	w	o
54	77	6F	20	4F	6E	65	20	4E	69	6E	65	20	54	77	6F

Plaintext in Hex (128 bits): 54 77 6F 20 4F 6E 65 20 4E 69 6E 65 20 54 77 6F

AES Example - The first Roundkey

- Key in Hex (128 bits): 54 68 61 74 73 20 6D 79 20 4B 75 6E 67 20 46 75
- $w[0] = (54, 68, 61, 74), w[1] = (73, 20, 6D, 79), w[2] = (20, 4B, 75, 6E), w[3] = (67, 20, 46, 75)$
- $g(w[3])$:
  - circular byte left shift of $w[3]$: $(20, 46, 75, 67)$
  - Byte Substitution (S-Box): $(B7, 5A, 9D, 85)$
  - Adding round constant $(01, 00, 00, 00)$ gives: $g(w[3]) = (B6, 5A, 9D, 85)$
- $w[4] = w[0] \oplus g(w[3]) = (E2, 32, FC, F1)$:

0101 0100	0110 1000	0110 0001	0111 0100
1011 0110	0101 1010	1001 1101	1000 0101
1110 0010	0011 0010	1111 1100	1111 0001
E2	32	FC	F1

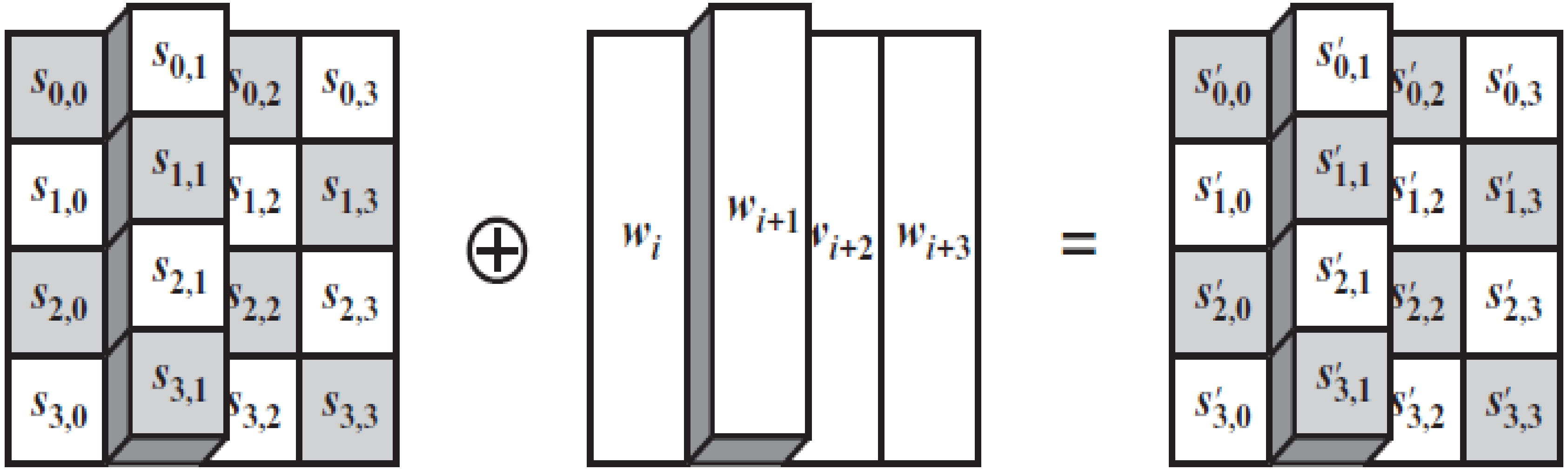
- $w[5] = w[4] \oplus w[1] = (91, 12, 91, 88)$, $w[6] = w[5] \oplus w[2] = (B1, 59, E4, E6)$,
  $w[7] = w[6] \oplus w[3] = (D6, 79, A2, 93)$
- first roundkey: E2 32 FC F1 91 12 91 88 B1 59 E4 E6 D6 79 A2 93

AES Example - All RoundKeys

- Round 0: 54 68 61 74 73 20 6D 79 20 4B 75 6E 67 20 46 75
- Round 1: E2 32 FC F1 91 12 91 88 B1 59 E4 E6 D6 79 A2 93
- Round 2: 56 08 20 07 C7 1A B1 8F 76 43 55 69 A0 3A F7 FA
- Round 3: D2 60 0D E7 15 7A BC 68 63 39 E9 01 C3 03 1E FB
- Round 4: A1 12 02 C9 B4 68 BE A1 D7 51 57 A0 14 52 49 5B
- Round 5: B1 29 3B 33 05 41 85 92 D2 10 D2 32 C6 42 9B 69
- Round 6: BD 3D C2 B7 B8 7C 47 15 6A 6C 95 27 AC 2E 0E 4E
- Round 7: CC 96 ED 16 74 EA AA 03 1E 86 3F 24 B2 A8 31 6A
- Round 8: 8E 51 EF 21 FA BB 45 22 E4 3D 7A 06 56 95 4B 6C
- Round 9: BF E2 BF 90 45 59 FA B2 A1 64 80 B4 F7 F1 CB D8
- Round 10: 28 FD DE F8 6D A4 24 4A CC C0 A4 FE 3B 31 6F 26

AddRoundKey

In this operation, the 128 bits of **State** are bitwise XORed with the 128 bits of the round key. Here is an example where the first matrix is State, and the second matrix is the round key.



$$\begin{pmatrix} 54 & 4F & 4E & 20 \\ 77 & 6E & 69 & 54 \\ 6F & 65 & 6E & 77 \\ 20 & 20 & 65 & 6F \end{pmatrix} \oplus \begin{pmatrix} 54 & 73 & 20 & 67 \\ 68 & 20 & 4B & 20 \\ 61 & 6D & 75 & 46 \\ 74 & 79 & 6E & 75 \end{pmatrix} = \begin{pmatrix} 00 & 3C & 6E & 47 \\ 1F & 4E & 22 & 74 \\ 0E & 08 & 1B & 31 \\ 54 & 59 & 0B & 1A \end{pmatrix}$$

e.g., $69 \oplus 4B = 22$

$$\begin{array}{r} 0110\ 1001 \\ \underline{0100\ 1011} \\ 0010\ 0010 \end{array}$$

SubBytes

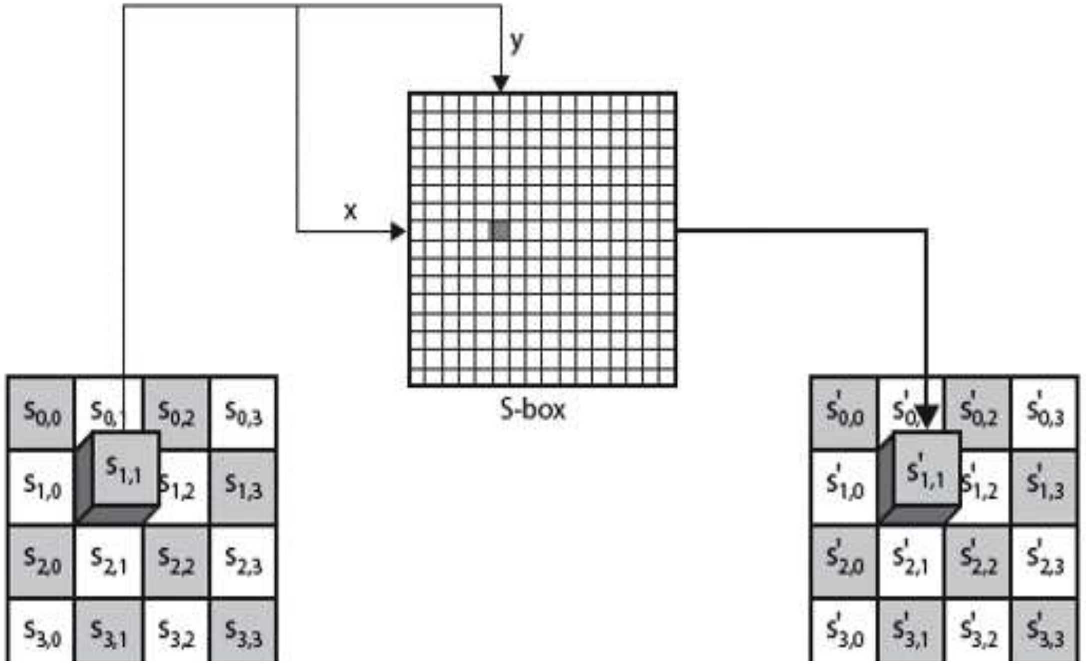
•**SubBytes** is a **nonlinear substitution** step.
•Each byte in the **state matrix** is replaced using the **AES S-Box**.

e.g.:

$$\text{state} = \begin{pmatrix} 00 & 3C & 6E & 47 \\ 1F & 4E & 22 & 74 \\ 0E & 08 & 1B & 31 \\ 54 & 59 & 0B & 1A \end{pmatrix} \Rightarrow \text{S\_box(State)} = \begin{pmatrix} 63 & EB & 9F & A0 \\ C0 & 2F & 93 & 92 \\ AB & 30 & AF & C7 \\ 20 & CB & 2B & A2 \end{pmatrix}$$

•The byte (e.g., 6E) is split into:
- •**Row** = 6 (first 4 bits)
- •**Column** = E (last 4 bits)

•The substitution value is taken from **S-Box[6][E]** = 9F.



ShiftRows

$$
\begin{array}{|c|c|c|c|}
\hline
a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ \hline
a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ \hline
a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ \hline
a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \\ \hline
\end{array}
\;\Rightarrow\;
\begin{array}{|c|c|c|c|}
\hline
a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ \hline
a_{1,1} & a_{1,2} & a_{1,3} & a_{1,0} \\ \hline
a_{2,2} & a_{2,3} & a_{2,0} & a_{2,1} \\ \hline
a_{3,3} & a_{3,0} & a_{3,1} & a_{3,2} \\ \hline
\end{array}
$$

no shift

one position left shift

one position left shift

one position left shift

e.g.:

$$
\begin{pmatrix}
63 & EB & 9F & A0 \\
C0 & 2F & 93 & 92 \\
AB & 30 & AF & C7 \\
20 & CB & 2B & A2
\end{pmatrix}
\Rightarrow
\begin{pmatrix}
63 & EB & 9F & A0 \\
2F & 93 & 92 & C0 \\
AF & C7 & AB & 30 \\
A2 & 20 & CB & 2B
\end{pmatrix}
$$

MixColumns

a linear mixing operation which multiplies fixed matrix against current State Matrix:

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

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

Unlike standard matrix multiplication, MixColumns performs matrix multiplication as per Galois Field (2^8).

e.g.:

$$\begin{pmatrix} 02\,03\,01\,01 \\ 01\,02\,03\,01 \\ 01\,01\,02\,03 \\ 03\,01\,01\,02 \end{pmatrix}\begin{pmatrix} 63 & EB & 9F & A0 \\ 2F & 93 & 92 & C0 \\ AF & C7 & AB & 30 \\ A2 & 20 & CB & 2B \end{pmatrix} = \begin{pmatrix} BA & 84 & E8 & 1B \\ 75 & A4 & 8D & 40 \\ F4 & 8D & 06 & 7D \\ 7A & 32 & 0E & 5D \end{pmatrix}$$

AES Example - Add Roundkey, Round 1

- State Matrix and Roundkey No.1 Matrix:

$$\begin{pmatrix} BA & 84 & E8 & 1B \\ 75 & A4 & 8D & 40 \\ F4 & 8D & 06 & 7D \\ 7A & 32 & 0E & 5D \end{pmatrix} \qquad \begin{pmatrix} E2 & 91 & B1 & D6 \\ 32 & 12 & 59 & 79 \\ FC & 91 & E4 & A2 \\ F1 & 88 & E6 & 93 \end{pmatrix}$$

- XOR yields new State Matrix

$$\begin{pmatrix} 58 & 15 & 59 & CD \\ 47 & B6 & D4 & 39 \\ 08 & 1C & E2 & DF \\ 8B & BA & E8 & CE \end{pmatrix}$$

- AES output after Round 1: 58 47 08 8B 15 B6 1C BA 59 D4 E2 E8 CD 39 DF CE

AES Example - Round 10

- after Substitute Byte and after Shift Rows:

$$\begin{pmatrix} 01 & 3A & 8C & 21 \\ 33 & 3E & B0 & E2 \\ 3D & B8 & 8E & 04 \\ BC & 4D & 1C & A7 \end{pmatrix} \qquad \begin{pmatrix} 01 & 3A & 8C & 21 \\ 3E & B0 & E2 & 33 \\ 8E & 04 & 3D & B8 \\ A7 & BC & 4D & 1C \end{pmatrix}$$

- after Roundkey (Attention: no Mix columns in last round):

$$\begin{pmatrix} 29 & 57 & 40 & 1A \\ C3 & 14 & 22 & 02 \\ 50 & 20 & 99 & D7 \\ 5F & F6 & B3 & 3A \end{pmatrix}$$

- ciphertext: 29 C3 50 5F 57 14 20 F6 40 22 99 B3 1A 02 D7 3A

AES Block Cipher

The AES Decryption Algorithm:

❑ AddRoundKey:

Add Roundkey transformation is identical to the forward add round key transformation, because the XOR operation is its own inverse.

❑ Inverse SubBytes:

This operation can be performed using the inverse S-Box. It is read identically to the S-Box matrix.

❑ InvShiftRows:

Inverse Shift Rows performs the circular shifts in the opposite direction for each of the last three rows, with a one-byte circular right shift for the second row, and so on.

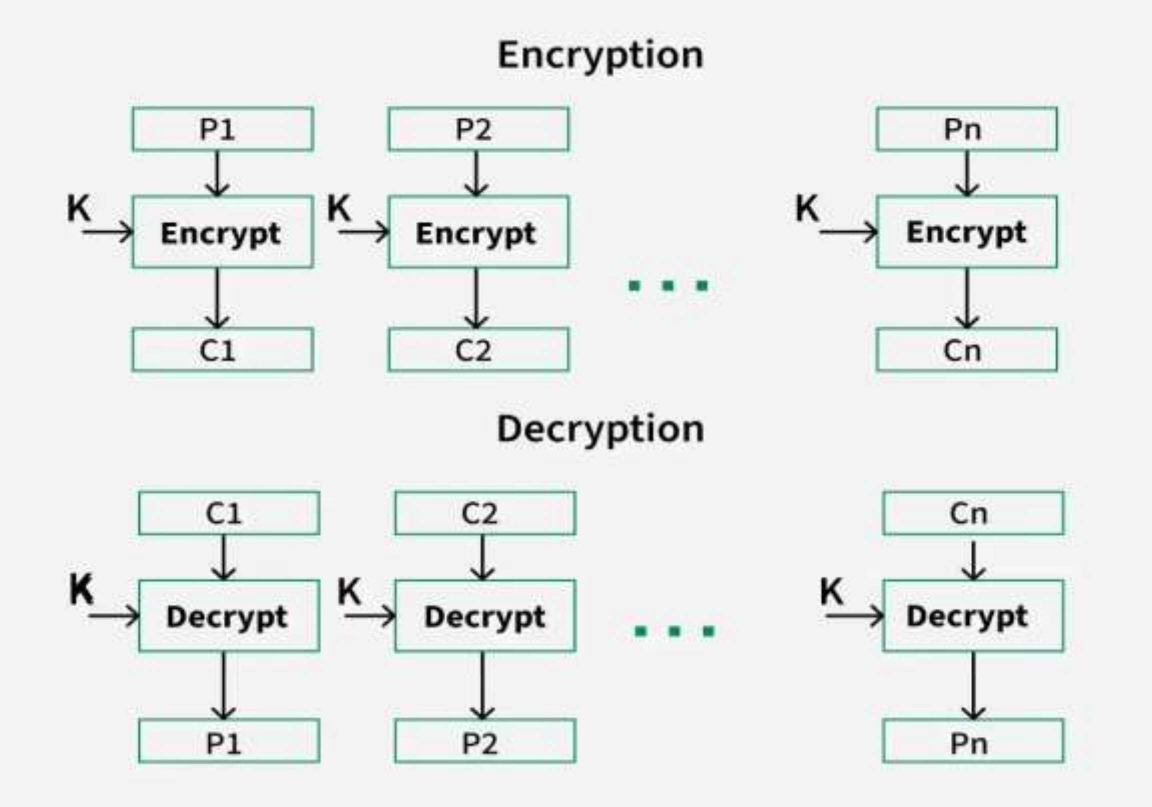
❑ InvMixColumns:

The inverse mix column transformation is defined by the following matrix multiplication in Galois Field (2^8):

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix}$$

Electronic Code Book (ECB)

- Simplest block cipher mode.
- Each block encrypted **independently**



Let e() be a block cipher of block size b, and let xi and yi be bit strings of length b:

Encryption:

$$y_1 = e_k(x_i) \quad i \geq 1$$

Decryption:

$$x_i = e\_k^{\{-1\}}(y_i) \quad i \geq 1$$

Cipher block chaining mode (CBC):

Converts block cipher into self-synchronizing stream cipher.
Encrypts IV, then XORs with plaintext

Let e() be a block cipher of block size b; let x_i and y_i be bit strings of length b; and IV be a nonce of length b.

Encryption(first block):

$$y_1 = e_k(x_i \oplus IV) \qquad i \geq 1$$

Encryption (general block):

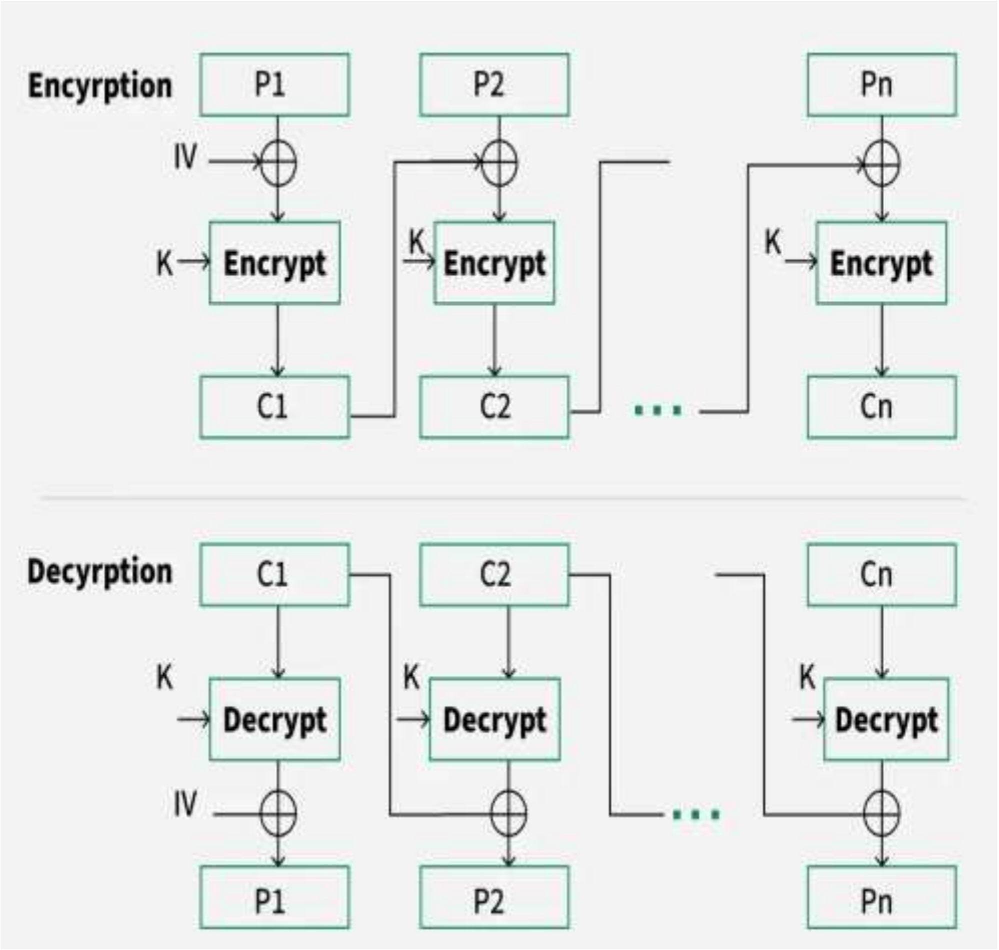
$$y_1 = e_k(x_i \oplus y_{i-1}) \qquad i \geq 2$$

Decryption (first block):

$$x_1 = e_k^{\{-1\}} \oplus IV$$

Decryption (general block):

$$x_i = e_k^{\{-1\}} xor \; y_{i-1} \qquad i>=2$$



Cipher feedback mode (CFB):

•Each plaintext block is XORed with the previous **ciphertext block.**
•**IV (Initialization Vector)** is used for the first block.

Let e() be a block cipher of block size b; let x_i and y_i be bit strings of length b; and IV be a nonce of length b.

Encryption(first block):

$$y_1 = e_k(IV)\ xor\ x_1$$

Encryption (general block):

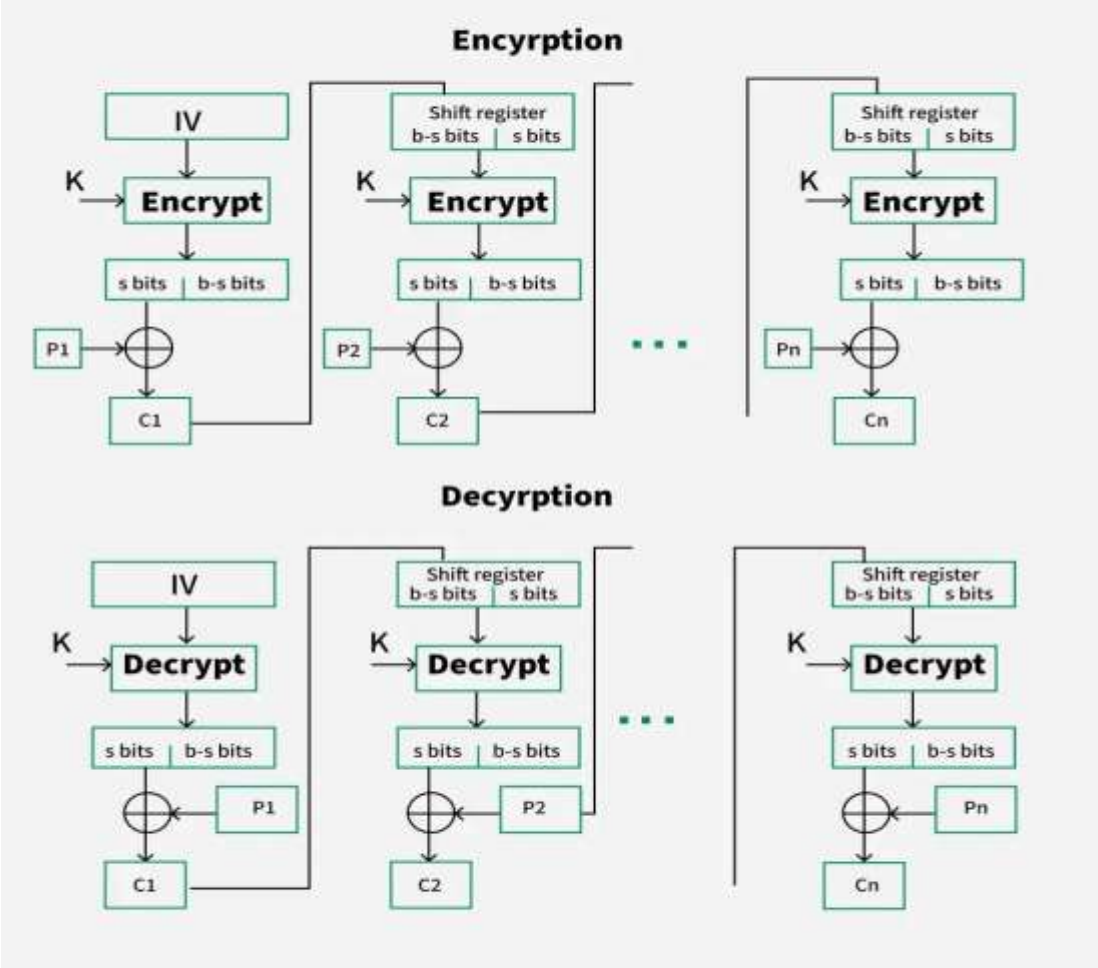
$$y_i = e_k(y_{i-1}) xor\ x_i \qquad i \geq 2$$

Decryption (first block):

$$x_1 = e_k(IV)\ xor\ y_1$$

Decryption (general block):

$$x_i = e_k(y_{i-1}) xor\ y_i \qquad i \geq 2$$



Output feedback mode (OFB):

Uses encrypted output as feedback instead of ciphertext.
Entire block output is used, making it a stream-like cipher.

Let e() be a block cipher of block size b; let x_i and y_i and s_i be bit strings of length b; and IV be a nonce of length b.

Encryption(first block):

$$s_1 = e_k(IV) \; and \;\; y_1 = s_1 xor \; x_1$$

Encryption (general block):

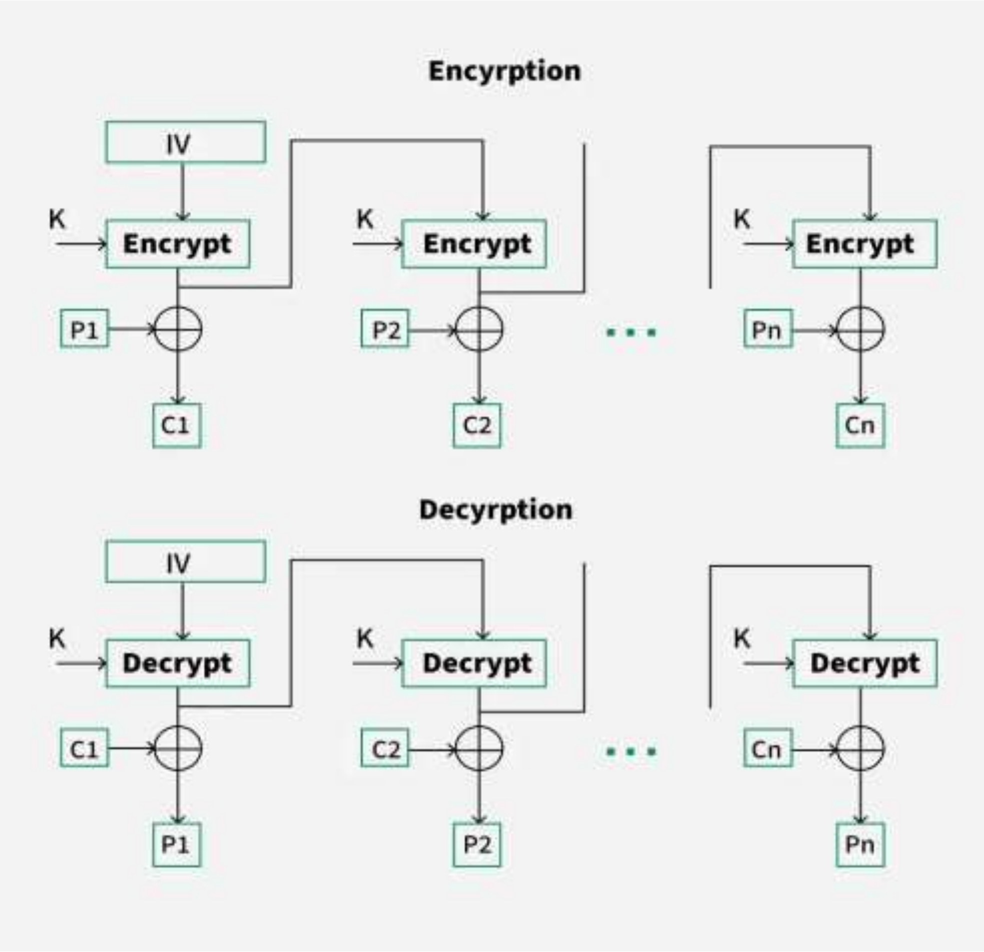
$$s_i = e_k(y_{i-1}) \; and \; y_i = s_i xor \; x_i \qquad i \geq 2$$

Decryption (first block):

$$s_1 = e_k(IV) and \; x_1 = s\_1 \; xor \; y_1$$

Decryption (general block):

$$s_i = e_k(y_{i-1}) \;\; and \; x_i = s_i xor \; y_i \qquad i \geq 2$$



Counter mode (CTR):

Encrypts a **counter** for each block.
Counter is incremented for each block.

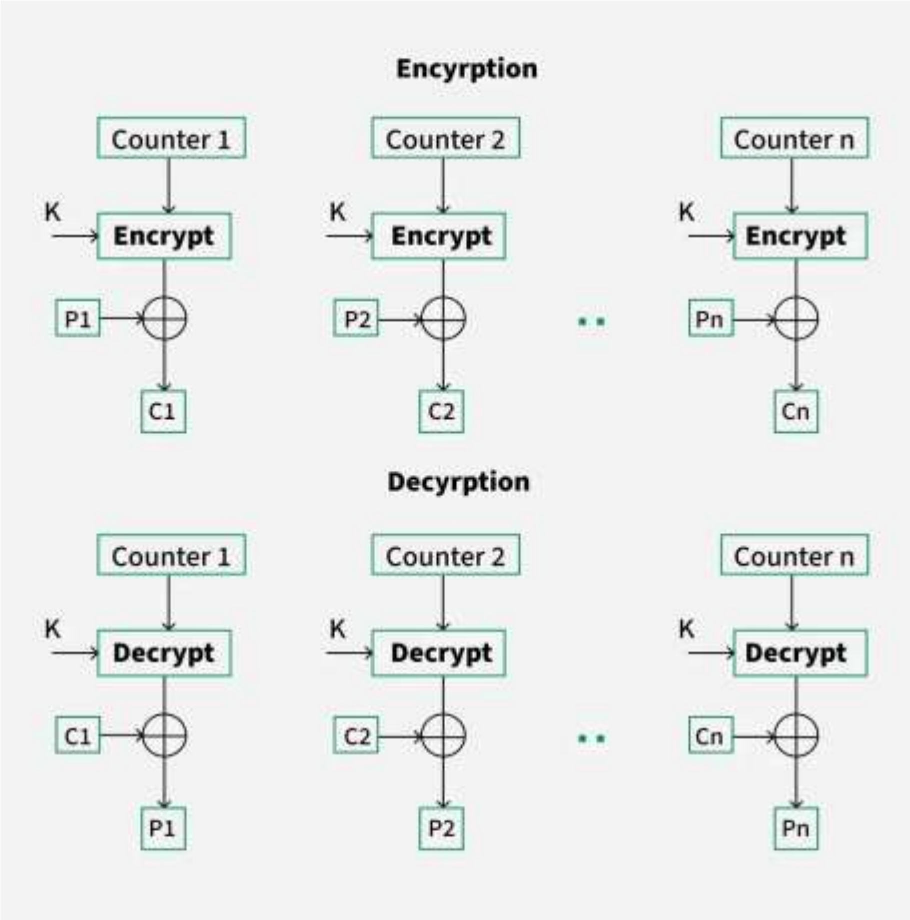
Let e() be a block cipher of block size b, and let xi and yi be bit strings of length b. The concatenation of the initialization value IV and the counter CT Ri is denoted by (IV||CTRi) and is a bit string of length b

Encryption:

$$y_i = e_k(IV||ctr_i) xor\ x_i \quad i \geq 1$$

Decryption

$$x_i = e_k(IV||ctr_i) xor\ y_i \quad i \geq 1$$



Galois Counter Mode (GCM)

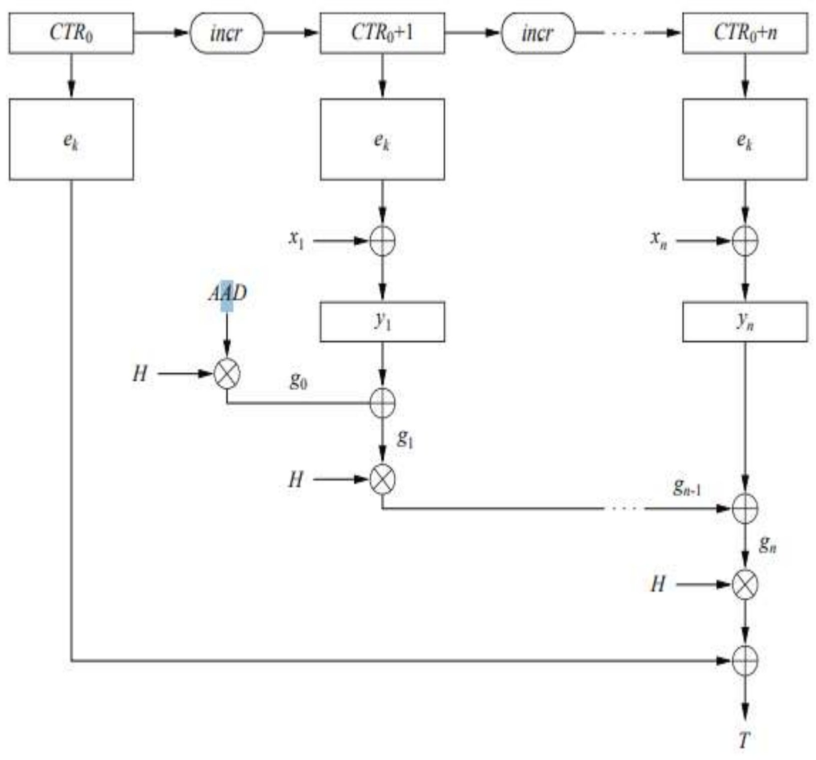
Combines **CTR mode + Authentication (via GHASH).**
Provides **confidentiality + integrity.**
Used In: TLS, VPNs, IPsec.

Let e() be a block cipher of block size 128 bit; let x be the plaintext consisting of the blocks x1,...,xn; and let AAD be the additional authenticated data.

Encryption(first block):
Derive a counter value CT R0 from the IV and compute
CTR1 = CTR0 +1.
Compute ciphertext:

$$y_i = e_k(CTR_i)\ xor\ x_i \quad i \geq 1$$

. Authentication a. Generate authentication subkey H = ek(0)
b. Compute g0 = AAD×H (Galois field multiplication)
c. Compute $g_i = (g_{i-1}\ xor\ y_i)\ \times H \quad 1 \leq i \leq n$
d. Final authentication tag: $T = (g_n \times H) xor\ \ e_k(CTR_o)$



References:

- https://csrc.nist.gov/archive/aes/rijndael/Rijndael-ammended.pdf

- Paar, C., & Pelzl, J. (2010). *Understanding Cryptography: A Textbook for Students and Practitioners*. Springer.

- **Schinta.** (n.d.). *Blowfish Encryption Algorithm* [PDF]. CS Department, Indiana State University. Available at: https://cs.indstate.edu/~schinta/blowfish.pdf

- **Tita, F., Setiawan, A., & Susanto, B.**
  Construction of Substitution Box (S-Box) Based on Irreducible Polynomials on GF(2^8).

Thank you