The Advanced Encryption Standard (AES) is a symmetric encryption algorithm widely used for securing data. AES encrypts data in blocks of 128 bits using keys of lengths 128, 192, or 256 bits.

Key Features of AES

Symmetric Encryption: The same key is used for both encryption and decryption.

Block Cipher: Operates on fixed-size blocks of data (128 bits).

Key Sizes: Supports 128-bit, 192-bit, and 256-bit keys.

Rounds:

AES-128: 10 rounds

AES-192: 12 rounds

AES-256: 14 rounds

Security: Resistant to most known attacks when used properly.

**AES Encryption Process**

AES operates on a 4x4 grid of bytes called the state. For simplicity, let's work with AES-128 (10 rounds, 128-bit key).

Example

Plaintext: 00112233445566778899aabbccddeeff

Key: 000102030405060708090a0b0c0d0e0f

Steps to Solve AES Encryption

1. Convert Plaintext and Key to Matrices

Represent the plaintext and the key as 4x4 matrices in column-major order:

Plaintext Matrix:

[00 44 88 cc]

[11 55 99 dd]

[22 66 aa ee]

[33 77 bb ff]

Key Matrix:

[00 04 08 0c]

[01 05 09 0d]

[02 06 0a 0e]

[03 07 0b 0f]

2. Key Expansion

Use the key schedule algorithm to generate 10 round keys. The key expansion involves:

* Rotating the last column.
* Substituting bytes using the S-Box.
* XORing with constants (Rcon) and previous columns.

For this example, calculate the first two round keys:

Round Key 1 (same as original key for AES-128):

[00 04 08 0c]

[01 05 09 0d]

[02 06 0a 0e]

[03 07 0b 0f]

3. Initial AddRoundKey

* XOR the plaintext matrix with the first round key:

Result:

[00 ⊕ 00 = 00, 44 ⊕ 04 = 40, 88 ⊕ 08 = 80, cc ⊕ 0c = c0]

[11 ⊕ 01 = 10, 55 ⊕ 05 = 50, 99 ⊕ 09 = 90, dd ⊕ 0d = d0]

[22 ⊕ 02 = 20, 66 ⊕ 06 = 60, aa ⊕ 0a = a0, ee ⊕ 0e = e0]

[33 ⊕ 03 = 30, 77 ⊕ 07 = 70, bb ⊕ 0b = b0, ff ⊕ 0f = f0]

Main Rounds (9 Rounds for AES-128)

Each Round Includes:

- **SubBytes**: Substitute each byte in the matrix using the S-Box. For example:

Byte 00 → 63 (from S-Box)

Byte 40 → f2

Byte 80 → 30

Perform this for all 16 bytes in the matrix.

- **ShiftRows**:

Rotate rows of the state:

Row 0: No change.

Row 1: Shift left by 1 byte.

Row 2: Shift left by 2 bytes.

Row 3: Shift left by 3 bytes.

- **MixColumns** (Skip in the final round):

* Multiply each column of the matrix by a fixed polynomial:

Matrix multiplication (GF(2^8)):

New column = [2 3 1 1] ⊗ Old column

- **AddRoundKey**:

XOR the current state matrix with the round key.

5. Final Round (10th Round)

* Perform **SubBytes, ShiftRows, and AddRoundKey** (skip MixColumns).

6. Ciphertext

The final matrix is the ciphertext:

Ciphertext: [xx xx xx xx]

[xx xx xx xx]

[xx xx xx xx]

[xx xx xx xx]

**Decryption Process**

For decryption, reverse the steps:

1. Inverse Key Expansion:

Generate round keys in reverse order.

2. Initial AddRoundKey:

* XOR the ciphertext with the last round key.

3. Inverse Rounds:

Perform the reverse of each step:

* **InvShiftRows**: Reverse the row shifts.
* **InvSubBytes**: Use the inverse S-Box.
* **InvMixColumns**: Multiply by the inverse polynomial matrix.

4. Final Round:

* Perform InvShiftRows, InvSubBytes, and AddRoundKey (no InvMixColumns).

5. Plaintext:

* The final matrix is the original plaintext.

Manual Tools for Solving

* S-Box Table: Used for substitution during SubBytes.
* Rcon Table: Used for key expansion.
* Polynomial Multiplication: For MixColumns and InvMixColumns.
* XOR Tables: To calculate AddRoundKey.

Example Reference

If you'd like to manually work through a small example,

start with the plaintext 00112233445566778899aabbccddeeff and

the key 000102030405060708090a0b0c0d0e0f. Follow the steps described, and use tables like the AES S-Box to look up values.

Sample code

**Bash**

pip install pycryptodome

from Crypto.Cipher import AES

from Crypto.Util.Padding import pad, unpad

import os

# Function to generate a random AES key

def generate\_key(key\_size=16):

return os.urandom(key\_size)

# Function to encrypt data

def encrypt\_aes(plaintext, key):

# Create a cipher object with AES in CBC mode

cipher = AES.new(key, AES.MODE\_CBC)

# Pad the plaintext to ensure it's a multiple of the block size

ciphertext = cipher.encrypt(pad(plaintext.encode(), AES.block\_size))

return cipher.iv, ciphertext

# Function to decrypt data

def decrypt\_aes(iv, ciphertext, key):

# Create a cipher object with the same key and IV

cipher = AES.new(key, AES.MODE\_CBC, iv)

# Decrypt and unpad the plaintext

plaintext = unpad(cipher.decrypt(ciphertext), AES.block\_size)

return plaintext.decode()

# Example usage

if \_\_name\_\_ == "\_\_main\_\_":

# Input data

plaintext = "Hello, AES Cipher!"

key = generate\_key() # Generate a 16-byte (128-bit) key

# Encrypt the plaintext

iv, ciphertext = encrypt\_aes(plaintext, key)

print("Original Plaintext:", plaintext)

print("Key (hex):", key.hex())

print("IV (hex):", iv.hex())

print("Ciphertext (hex):", ciphertext.hex())

# Decrypt the ciphertext

decrypted\_text = decrypt\_aes(iv, ciphertext, key)

print("Decrypted Plaintext:", decrypted\_text)