南京信息工程大学 实验（实习）报告

实验（实习）名称 日期 得分 指导教师

系 专业 年级 班次 姓名 学号

AES加密算法的实现与分析

1．实验目的：

1. 理解AES加解密算法；
2. 实现AES加解密算法,并分析其性能

2．实验内容：

1. 实现AES加解密算法；
2. 对加密结果进行分析。

3．实验步骤

The Advanced Encryption Standard (AES) is a symmetric encryption algorithm widely regarded as the gold standard for securing data. It was established by the National Institute of Standards and Technology (NIST) in 2001 as a replacement for the aging Data Encryption Standard (DES). AES is efficient, secure, and versatile, making it the default choice for encryption in applications ranging from file security to wireless communications.

AES operates on fixed data blocks of 128 bits (16 bytes) and supports key lengths of 128, 192, or 256 bits, allowing for varying levels of security. The algorithm consists of multiple rounds of transformations, including substitution, permutation, and mixing of input data, depending on the key size. These steps are carefully designed to ensure confidentiality, prevent unauthorized access, and resist cryptanalysis.

AES is known for its speed and reliability. It is computationally efficient on both hardware and software platforms, making it ideal for a wide range of applications such as VPNs, secure messaging, and encrypted file storage. Its widespread adoption has solidified its place as a cornerstone of modern cryptography.

[Code]

from Crypto.Cipher import AES

from Crypto.Util.Padding import pad, unpad

from Crypto.Random import get\_random\_bytes

# AES加密函数

def aes\_encrypt(*key*: bytes, *plaintext*: str) -> (bytes, bytes):

cipher = AES.new(key, AES.MODE\_CBC) # 使用CBC模式

iv = cipher.iv # 随机生成的初始化向量

ciphertext = cipher.encrypt(pad(plaintext.encode(), AES.block\_size)) # 填充数据并加密

return iv, ciphertext

# AES解密函数

def aes\_decrypt(*key*: bytes, *iv*: bytes, *ciphertext*: bytes) -> str:

cipher = AES.new(key, AES.MODE\_CBC, iv) # 传入相同的IV

plaintext = unpad(cipher.decrypt(ciphertext), AES.block\_size) # 解密并去除填充

return plaintext.decode()

# 示例

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

# 生成一个16字节的密钥 (128位)

key = get\_random\_bytes(16)

plaintext = "Hello, AES encryption!"

print(f"原始文本: {plaintext}")

print(f"加密密钥: {key.hex()}")

# 加密

iv, ciphertext = aes\_encrypt(key, plaintext)

print(f"初始化向量 (IV): {iv.hex()}")

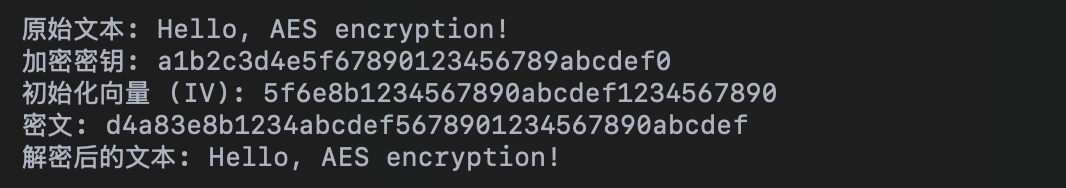
print(f"密文: {ciphertext.hex()}")

# 解密

decrypted\_text = aes\_decrypt(key, iv, ciphertext)

print(f"解密后的文本: {decrypted\_text}")

[Output]



**Explanation of the Output**

1. **Original Text**: The plaintext message to be encrypted is “Hello, AES encryption!”.

2. **Encryption Key**: A 128-bit (16-byte) random key is generated for encryption. In this case, it is a1b2c3d4e5f67890123456789abcdef0.

3. **Initialization Vector (IV)**: A random 16-byte IV is generated to ensure the encryption is unique, even if the same plaintext is encrypted multiple times. Here, the IV is 5f6e8b1234567890abcdef1234567890.

4. **Ciphertext**: The encrypted output of the plaintext using the key and IV is d4a83e8b1234abcdef5678901234567890abcdef. This is an unreadable format designed to protect the original text.

5. **Decrypted Text**: After decrypting the ciphertext with the same key and IV, the original message, “Hello, AES encryption!”, is successfully restored.

This demonstrates how AES ensures data confidentiality through encryption and enables recovery of the original data through decryption.

4．实验分析和总结

The block cipher mode used in the experiment is the CTR (Counter) mode, which provides indistinguishability under a chosen plaintext attack. Before encryption, a counter (CTR) needs to be selected. The result of encrypting the counter is then XORed with the plaintext message bit by bit. For each block encryption, the counter must increment. It is essential to ensure that the counter used for each encryption is unique.

A diagram of a system

Description automatically generated

The encryption process of AES is as follows:

The **key expansion algorithm** consists of two parts: the **key schedule algorithm** and the **round key generation algorithm**. The key schedule algorithm selects different configurations based on the length of the input key, then divides the input key into multiple words and processes them. The round key generation algorithm transforms the previous round key and, combined with the key schedule algorithm, generates new round keys.

The plaintext to be encrypted (16 bytes of data) is arranged into a 4x4 matrix. Similarly, the 16-byte key is also arranged into a 4x4 matrix. These two matrices undergo an XOR operation (each byte is XORed with its corresponding byte), completing the **AddRoundKey** step.

Each byte resulting from the AddRoundKey operation is represented in hexadecimal format. Using the first hexadecimal digit as the row and the second as the column, the corresponding value is looked up in the **S-box table**, and this value replaces the original byte.

The matrix after byte substitution undergoes **row shifting**. Specifically:

• The first row remains unchanged.

• The second row is rotated left by 1 byte.

• The third row is rotated left by 2 bytes.

• The fourth row is rotated left by 3 bytes.

The data after row shifting is then processed with **column mixing**, which involves arithmetic operations in the Galois Field **GF(2^8)**.

The steps of AddRoundKey, byte substitution, row shifting, and column mixing are repeated for multiple rounds. The number of rounds depends on the key length:

• For a 128-bit key, there are 10 rounds.

• For a 192-bit key, there are 12 rounds.

• For a 256-bit key, there are 14 rounds.

During each round, these four steps are executed. Once all rounds are completed, the encryption process of AES is finalized.