AES encryption

> This section is relatively complex and requires a certain theoretical foundation. Skipping this section will not affect the subsequent content.
>
> The source code of this section is located in [Source Code Summary/02.Basic/12.aes.py]

What is AES?

AES (Advanced Encryption Standard) is a symmetric encryption algorithm that is widely used to protect sensitive data. In 2001, the National Institute of Standards and Technology (NIST) of the United States established it as the standard encryption method, replacing the previous DES (Data Encryption Standard).

Basic Features of AES

- **Symmetric encryption** : the same key is used for encryption and decryption
- **Block cipher** : Encryption in fixed-length blocks
- **Key length** : Supports 128-bit, 192-bit and 256-bit key lengths
- **Packet size** : fixed at 128 bits (16 bytes)

AES working mode

AES has several operating modes, the most common ones include:

1. **ECB (Electronic Codebook)** : The simplest mode, directly encrypting each data block independently
2. **CBC (Cipher Block Chaining)** : Each plaintext block is first XORed with the previous ciphertext block and then encrypted
3. **CTR (Counter)** : Use the counter to generate the key stream and XOR it with the plaintext
4. **GCM (Galois/Counter Mode)** : Combines with CTR mode and provides authentication function

Code Analysis

Let's analyze the example code, which uses `ucryptolib` the module in MicroPython to demonstrate AES encryption:

1. Importing modules

```
import  ucryptolib
```

The cryptographic library in MicroPython is imported here `ucryptolib`, which is a lightweight cryptographic library designed for resource-constrained devices.

2. AES-128 ECB mode test

```
key = b'1234567890abcdef'   # 16-byte key
plaintext = b'This is 16 bytes'   # 16 bytes of plaintext
```

Here it is defined:

- **Key** : 16 bytes in length, corresponding to AES-128
- **Plaintext** : The data to be encrypted, exactly 16 bytes

```
cipher = ucryptolib . aes ( key , ucryptolib . MODE_ECB )
encrypted = cipher . encrypt ( plaintext )
```

These two lines of code:

1. Create an AES cipher object using ECB mode
2. Encrypt the plaintext to get the ciphertext

```
cipher = ucryptolib . aes ( key , ucryptolib . MODE_ECB )
decrypted = cipher . decrypt ( encrypted )
```

These two lines of code:

1. Create a new AES object (for decryption)
2. Decrypt the ciphertext to get the decrypted data

```
assert  decrypted == plaintext , "AES-128 ECB failed"
```

This line of code verifies that the decrypted data is the same as the original plaintext, and reports an error if they are different.

3. Invalid key length test

```
try :
    ucryptolib . aes ( b'short_key' , ucryptolib . MODE_ECB )
    assert  False , "Invalid key not detected"
except  ValueError :
    print ( "Passed" )
```

This code tests for the use of an incorrect key length. AES requires that the key length must be 16, 24, or 32 bytes, and here a shorter key is used, which would be expected to throw `ValueError` an exception.

4. Data length misalignment test

```
cipher = ucryptolib . aes ( key , ucryptolib . MODE_ECB )
try :
    cipher . encrypt ( b'short' )
    assert  False , "Unaligned data not detected"
except  ValueError :
    print ( "Passed" )
```

This code tests the case where the data length is not a multiple of 16 bytes. In ECB mode, the data length must be a multiple of 16 bytes. Here, a data length of only 5 bytes is used, and `ValueError` an exception is expected to be thrown.

AES key concepts explained

1. Key length

- **AES-128** : uses a 16-byte (128-bit) key
- **AES-192** : uses a 24-byte (192-bit) key
- **AES-256** : uses a 32-byte (256-bit) key

Longer keys provide greater security, but may slightly reduce performance.

2. Limitations of the ECB model

The sample code uses ECB mode, which is the simplest but also the least secure AES mode:

- The same plaintext block will produce the same ciphertext block
- Cannot hide data patterns, may leak information
- No Initialization Vector (IV) is used

For practical applications, it is recommended to use more secure modes such as CBC, CTR or GCM.

3. Filling problem

AES requires that the data length is a multiple of 16 bytes. When the data length does not meet the requirement, padding is required:

- **PKCS#7 padding** : fill with missing bytes (most commonly used)
- **Zero padding** : fill with 0
- **ANSI X.923** : The last byte indicates the number of bytes to be filled, and the rest are filled with 0

The sample code does not handle padding, but directly uses the data with a length of exactly 16 bytes.