SHA256 encryption

Introduction

SHA256 is a commonly used hash algorithm in cryptography and belongs to the SHA-2 (Secure Hash Algorithm 2) family. It can convert input data of any length into an output of fixed length (256 bits/32 bytes), which is usually represented by a 64-bit hexadecimal number.

Characteristics of SHA256

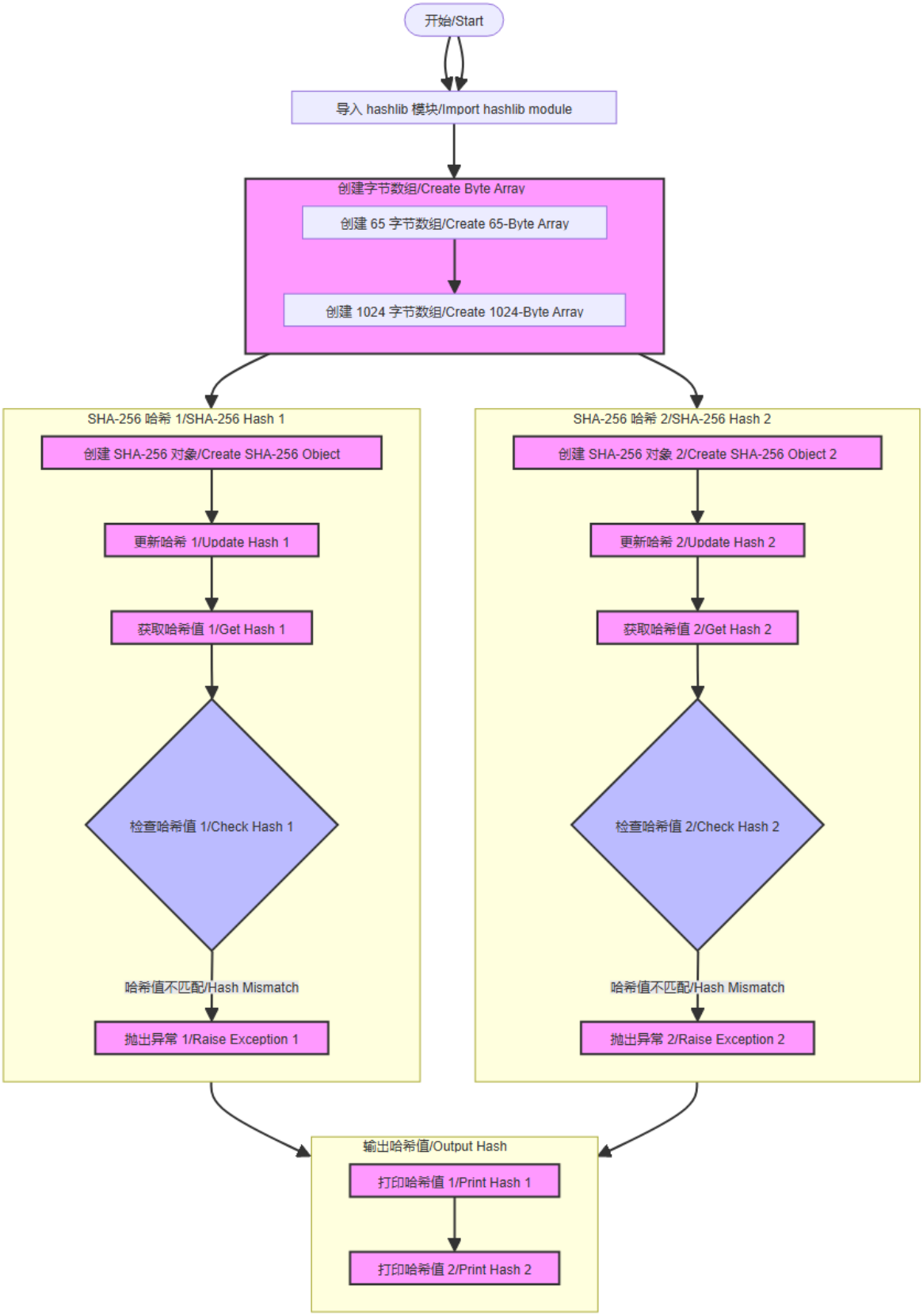
- **One-way** : It is difficult to deduce the input from the output
- **Determinism** : The same input always produces the same output
- **Avalanche effect** : small changes in input can lead to large differences in output
- **Collision resistance** : It is very difficult to find two different inputs that produce the same output
- **Fixed-length output** : 256 bits (32 bytes)

Application Scenario

1. Digital Signature
2. Data integrity verification
3. Password Storage
4. Blockchain Technology
5. Random Number Generation

Code Analysis

The routine code execution flow chart is as follows



The complete code is located in [Source Code/02.Basic/11.sha256.py]

```
# 导入 hashlib 模块，用于提供 SHA-256 哈希功能
# (Import the hashlib module for SHA-256 hashing functionality)
import hashlib

# 创建一个包含65个零字节的字节数组
# (Create a byte array containing 65 zero bytes)
```

```
a = bytes([0] * 65)

# 创建一个 SHA-256 哈希对象
# (Create a SHA-256 hash object)
b = hashlib.sha256()

# 更新哈希对象，使用字节数组 a 进行两次更新
# (Update the hash object with the byte array a twice)
b.update(a)
b.update(a)

# 获取最终的哈希值
# (Get the final hash value)
c = b.digest()

# 打印哈希值
# (Print the hash value)
print(c)

# 检查哈希值是否与预期值相同
# (Check if the hash value matches the expected value)
if c !=
b"\xe5Z\\'sj\x87a\xc8\xe9j\xce\xc0r\x10#%\xe0\x8c\xb2\xd0\xdb\xb4\xd4p,\xfe8\xf8
\xab\x07\t":
    # 如果不相同，则抛出异常并打印当前哈希值
    # (If not, raise an exception and print the current hash value)
    raise(Exception("error! {}".format(c)))

# 创建一个包含1024个零字节的字节数组
# (Create a byte array containing 1024 zero bytes)
a = bytes([0] * 1024)

# 使用字节数组 a 创建一个新的 SHA-256 哈希对象
# (Create a new SHA-256 hash object using the byte array a)
b = hashlib.sha256(a)

# 获取新的哈希值
# (Get the new hash value)
c = b.digest()

# 打印新的哈希值
# (Print the new hash value)
print(c)

# 检查新的哈希值是否与预期值相同
# (Check if the new hash value matches the expected value)
if c !=
b'_p\xbf\x18\xa0\x86\x00p\x16\xe9H\xb0J\xed;\x82\x10:6\xbe\xa4\x17U\xb6\xcd\xdf\
xaf\x10\xac\xe3\xc6\xef':
    # 如果不相同，则抛出异常并打印当前哈希值
    # (If not, raise an exception and print the current hash value)
    raise(Exception("error! {}".format(c)))
```

Code Key Points

1. **There are two ways to create a hash object** :
    - First create an empty object, then use `update()` the method to add data
    - Pass data directly when creating an object
2. **Incremental hashing** :
    - The first part of the code shows `update()` the use of the method, which can be called multiple times to add data
    - This is equivalent to computing the hash value of the data concatenated together (i.e. 130 zero bytes)
3. **Hash result acquisition** :
    - `digest()` The method returns a hash value in binary format
    - You can also use `hexdigest()` to get the hexadecimal string representation
4. **Hash verification** :
    - The code ensures that the algorithm is implemented correctly by comparing the generated hash value with the expected value
    - Demonstrates the deterministic nature of the SHA256 algorithm (the same input always produces the same output)
5. **Input size and hash value** :
    - Regardless of the input data size (65 bytes, 130 bytes, or 1024 bytes), the output is always 32 bytes

Although SHA256 is powerful, it should be used in combination with other techniques such as salting and key stretching in certain security scenarios (such as password storage).