实验：消息认证与Hash算法

程昊苏

山东财经大学
山东省区块链金融重点实验室

2024年10月

实验目的：

1. 学会MAC与MDC算法的原理
2. 用Python实现MAC与MDC的计算
3. 学会对明文与密文进行完整性检测
4. 在python中学会应用MD5、SHA1，SHA256，SM3等算法

实验环境：

1. Python3环境（VS Code、IDLE等）
2. Python 第三方库：random、hashlib、cryptography、gmssl库

消息认证码MAC

MAC也称为密码校验和，它由下述函数产生

$$MAC = H(m||k)$$

其中，m是一个变长的消息，k是收发双方共享的密钥，$H(m||k)$是定长的认证符（Authenticator）。

- 在实际中，发送者将消息m和此认证符一起发给接收者：

$$\{m, H(m||k)\}$$

- 接收者在收到消息m′之后，计算：$MAC' = H(m'||k)$
  - 比较$MAC' = MAC$?
  - 若相等，则说明消息未被篡改；
  - 若不等，则说明消息被改动了。

消息检测码MDC

MDC也称为篡改检测码，它由下述函数产生

$$MDC = H(m)$$

其中，m是一个变长的消息，$H(m)$是定长的杂凑函数值作为认证符。

- 在实际中，发送者将消息m和认证符一起发给接收者：

$$\{m, H(m)\}$$

- 接收者在收到消息m'之后，计算：$MDC' = H(m')$
  - 比较$MDC' = MDC$?
  - 若相等，则说明消息未被篡改；
  - 若不等，则说明消息被改动了。

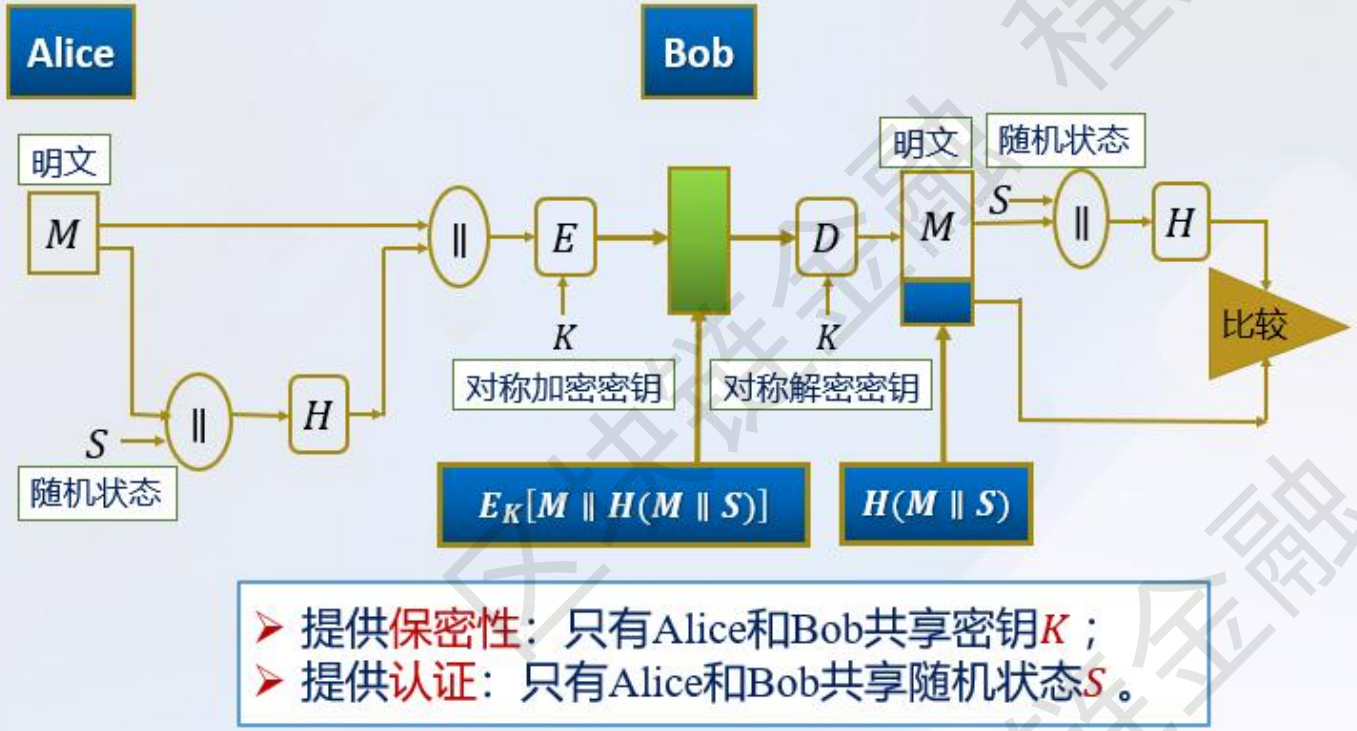
实验1：MDC

1. 新建MDC.py文件；
2. 使用python的 hashlib 三方库 ,参考：https://docs.python.org/3/library/hashlib.html
3. 使用input 获取用户输入的字符串，如 m1=“这里是明文消息 Hello World！”
4. 分别使用MD5、SHA1、SHA256、SHA512 计算 m的MDC值（hash.hexdigest()，用16进制表示）
5. 按照“MDC-算法名:MDC值”格式（如，MDC-SHA1:xxxxx），把上面字符串与算法的结果，使用UTF-8的编码（用“字符串”.encode(“UTF-8”)的方法）逐行保存到在“message1.txt”文件内（注意：明文应保存在文件的第一行）。
6. 用户重新输入一个字符串，如m2 = “这里是明文消息 Hello World”
7. 计算m2的MDC值并保存m2的明文和MDC值到“message2.txt”文件内
8. 观察对比两个文件内的明文m1，m2与对应的MDC值的变化（多种算法对应的hash值，说明每种算法的输出是多少位，观察雪崩效应）

实验2： MAC

1. 新建MAC.py文件；
2. 使用python的 hashlib 三方库 ,参考：https://docs.python.org/3/library/hashlib.html
3. 产生64位随机整数n，计算整数n的MDC值最为密钥k 使用(用k = SHA265(n), 注意：整数n需要转化成字节流后使用，如SHA265(str(n).encode())，（密钥k需要保护好，只能自己知道）
4. 从message1.txt中读取明文m，（按行读取第一行内容： m=“这里是明文消息 Hello World！“ ）
5. 分别使用MD5、SHA1、SHA256、SHA512 计算 m||k的MAC值（如，SHA1(m+k).hexdigest() ）
6. 按照“MAC1-算法名:MDC值”格式（如，MAC-SHA1:xxxxx），把上面字符串与算法的结果，追加保存到“message1.txt”文件内。
7. 使用新n2 =123456 , 得到新的密钥 k2= SHA256(str(n2).encode()).hexdigest(), 重复第5步内容，计算 m||k2的MAC值，并按“MAC2-算法名:MDC值”格式，追加保存到“message1.txt”文件内
8. 观察对比使用k1,k2 与同一明文m产生的MAC值的变化（注意对比多种算法的hash值的和其长度）

实验3： Hash算法的应用



1. 新建SM3andAES.py文件;
2. 构造H()、E()、D()、Comp()比较等函数
3. 分别建AliceDo()和BobDo()函数;
4. 分别在两个函数中完成相应的操作;
5. Alice加密后的内容输出到文件crypto.txt文件中，Bob读取该文件并做相应的后续操作。
6. 随机状态S 与 加密密钥K可以最为全局变量共享给Alice和Bob使用。

1. 分别用SM3，AES单钥加密算法实现上面的密码学协议的应用;
2. 明文M为实验2中生成的message1.txt文件，随机状态S为16位的随机整数，AES单钥加密与SM3算法参考后面PPT；生成密文后查看密文内容，改变密文内容后，让Bob验证，看是否还能通过验证。

参考： AES单钥算法的应用

1. 安装库：pip install cryptography
2. 使用库：import cryptography
3. 参考使用AES单钥加密算法，对明文m进行加解密；
4. 注意单钥加密算法的密钥key，只能由Alice和Bob双方知道，并由双方妥善保管。
5. 加密明文需要UTF-8的字节码 .encode（“UTF-8”）
6. 参考文档：
   https://cryptography.io/en/latest/

```python
import cryptography

from cryptography.fernet import Fernet

# 生成一个密钥
key = Fernet.generate_key()
print("使用的单钥加密密钥K：", key)

# 使用密钥创建一个Fernet对象
cipher_suite = Fernet(key)

# 需要加密的数据
message = "这是一个需要加密的消息".encode("UTF-8")

# 加密数据
encrypted_message = cipher_suite.encrypt(message)
print("加密后的消息：", encrypted_message)

# 解密数据
decrypted_message = cipher_suite.decrypt(encrypted_message)
print("解密后的消息：", decrypted_message.decode("UTF-8"))
```

参考： SM3算法的应用

1. 安装库： pip install gmssl
2. 使用库：from gmssl import sm3
3. 参考文档：
   https://pypi.org/project/gmssl/

sm3.py

```python
from gmssl import sm3


def sm3_hash(message: str):
    """
    国密sm3加密
    :param message: 消息值，bytes类型
    :return: 哈希值
    """

    msg_list = [i for i in bytes(message.encode('UTF-8'))]
    hash_hex = sm3.sm3_hash(msg_list)
    return hash_hex



# main
if __name__ == '__main__':
    message = "Hello Python!"
    print('message: {}\nsm3 hash: {}'.format(message, sm3_hash(message)))

```

Shell

```
>>> %Run sm3.py
  message: Hello Python!
  sm3 hash: 036191dd77ea6aae2b03e10527cd0fdbb75cdeeb035b5d7b4e39579a4425a01b

>>>
```