**高子恩202100460135**

**（一）**

\*Project1: implement the naïve birthday attack of reduced SM3

**实现方式**：python

**说明**：

它会生成随机的输入数据并计算其哈希值。

代码将使用一个哈希表来存储已计算过的哈希值和对应的输入数据。

当发现两个不同的输入数据具有相同的哈希值时，即发生碰撞，打印出这两个输入数据

**结果**：结果将类似于

1. Collision found!
2. Input 1: 123456
3. Input 2: 789012

**（三）**

\*Project3: implement length extension attack for SM3, SHA256, etc.

**实现方式**：python

**说明：**

基本思想嘛，是利用已知哈希值来扩展生成包含原始消息和附加数据的新消息的哈希值

我们首先计算原始消息的SHA-256哈希，并将该哈希作为输入，与附加数据一起调用长度扩展攻击函数。

**具体要点：**

# Assuming original\_hash is the SHA-256 hash of original\_message

# original\_length is the length of the original\_message in bits

# Calculate the padding required for the original\_message

# The state of the SHA-256 hash after processing the original\_message

# (the last 8 words of the SHA-256 intermediate hash)

# Initialize a new SHA-256 hash object

# Update the hash object with the additional\_data

# Calculate the length of the combined message (original\_message + padding + additional\_data)

# Update the internal state of the SHA-256 hash object with the length and padding

# Finalize the hash to get the length-extended hash

# Calculate the SHA-256 padding for a given message length in bits

**（四）**

\*Project4: do your best to optimize SM3 implementation (software)

**实现方式：**python，一个简化的SM3实现

**代码具体解释：**

1. **rotate\_left(x, n)**: 这是一个左循环移位函数，将一个32位整数x向左循环移动n位

2. **FF(X, Y, Z)**: 这是SM3算法中的一个逻辑函数，用于将三个32位整数X、Y和Z执行异或运算。

3. **GG(X, Y, Z)**: 另一个逻辑函数，用于将三个32位整数X、Y和Z执行与或运算。

4. **P0(X)`和`P1(X):** 这两个函数是SM3算法中的置换函数，通过使用rotate\_left函数对X进行位操作和异或运算来返回结果。

5. **T(j)**: 一个常量函数，用于在SM3算法的处理过程中提供常量值

6. **padding(msg)**: 实现SM3的消息填充规则，将输入消息msg进行填充，使其长度满足64字节的倍数，并添加了一个1和一些0来满足填充需求。

7. **SM3(message)**: 这是主要的SM3哈希函数实现。它接受输入消息message，并对其进行填充后，按照SM3算法的步骤对每个64字节（512位）的消息分组进行处理。最终得到一个长度为256位的哈希值。

**运行结果：**

比方说，用这个示例

message = b"hello, world!"

hash\_value = SM3(message)

print(hash\_value.hex())

结果将会是：

1. 0906a78ebbf4200d24cd02c5e13cb9bca884f3a56b57f388791b7955c7ff2eb8

**（五）**

\*Project5: Impl Merkle Tree following RFC6962

**实现方式**：C++

**说明**：

OpenSSL库中的SHA256哈希函数来计算数据的哈希值。然后，我们实现了一个MerkleTree类，其中buildTree方法用于构建Merkle树，getRootHash方法用于获取Merkle树的根哈希值。

在示例用法中，我们创建了一个包含四个证书的证书列表，并使用这些证书构建了Merkle树。最后，我们打印出证书列表和Merkle树的根哈希值。

**运行结果**：大概长这个样

1. 证书列表：
2. certificate1
3. certificate2
4. certificate3
5. certificate4
6. Merkle树根哈希值：49e4083b856d09729c477f4d1bf7388dd56b97cf5742ab0e77f5c98d21767964

**（八）**

\*Project8: AES impl with ARM instruction

**说明**：文档包含

1. AES impl with ARM步骤概述
2. 具体代码

（我给的代码只是一个简化的实现，没有包含完整的AES加密算法……）

**（九）**

\*Project9: AES / SM4 software implementation

**AES实现**：C++

**SM4实现**：C++（简化版本，实际SM4实现还需要MixColumns和KeyExpansion等操作）

**（十一）**

\*Project11: impl sm2 with RFC6979

**使用了：**

1. #include <openssl/bn.h>
2. #include <openssl/ec.h>
3. #include <openssl/obj\_mac.h>
4. #include <openssl/ecdsa.h>

**运行结果说明：**

1、签名的结果通常是一对大整数，表示r和s值

2、正确的签名结果应该能够通过验证过程成功验证

3、签名算法的运行时间和结果大小将取决于使用的曲线参数和消息的长度

**（十四）**

\*Project14: Implement a PGP scheme with SM2

1. #include <cryptopp/eccrypto.h>
2. #include <cryptopp/asn.h>
3. #include <cryptopp/oids.h>
4. #include <cryptopp/modes.h>
5. #include <cryptopp/filters.h>

文件为一个简化版本……

**（十七）**

\*Project17：比较Firefox和谷歌的记住密码插件的实现区别

**说明**：

从插件类型、API支持、功能实现三个方面说明

附上二者的开发者文档

**（二十二）**

\*Project22: research report on MPT

**要点：**

1、MPT用于有效地存储和检索键值对集合

2、MPT通过将键的前缀压缩在树的节点中，减少存储空间和访问时间。

3、MPT使用每个节点的哈希值来构建Merkle树，以验证数据是否被篡改。

4、在以太坊中，MPT用于存储账户状态和交易信息，使得区块链网络能够高效地处理交易和查询账户信息。