山东大学网络空间安全学院

创新创业实践 课程实验报告

学号：202100141038	姓名：沈文慧	班级：21 密码 2 班
实验题目：implement length extension attack for SHA256		
实验学时：	实验日期：2023-07	

问题分析：implement length extension attack for SHA256

直接运行 sha256_length_expanding_attack.py。根据 sha256 的特性，对 sha256 进行长度扩展攻击，由于 sha256 采用的也是 md 结构，所以进行长度扩展攻击的主要方式为首先得到一个之前的哈希值，之后再将新的消息内容添加在其后，计算出一个新的哈希值。对于正常的 sha256 算法而言，其初始的 h 值是固定的，会随着对每一个消息分块进行迭代后随之变化，这里需要做的就是对于 sha256 的实现进行一定的修改，使得其可以进入到任一轮次的迭代中。

硬件环境：Windows 11，X64

软件环境：visual studio code

实验步骤与内容：

1. 实验代码：

```
    #coding=utf-8
class SHA256:
    def __init__(self,h0):
        #64 个常量
        #图中 Kt
        self.constants = (
            0x428a2f98, 0x71374491, 0xb5c0fbcf, 0xe9b5dba5,
            0x3956c25b, 0x59f111f1, 0x923f82a4, 0xab1c5ed5,
            0xd807aa98, 0x12835b01, 0x243185be, 0x550c7dc3,
            0x72be5d74, 0x80deb1fe, 0x9bdc06a7, 0xc19bf174,
            0xe49b69c1, 0xefbe4786, 0x0fc19dc6, 0x240ca1cc,
            0x2de92c6f, 0x4a7484aa, 0x5cb0a9dc, 0x76f988da,
            0x983e5152, 0xa831c66d, 0xb00327c8, 0xbf597fc7,
            0xc6e00bf3, 0xd5a79147, 0x06ca6351, 0x14292967,
            0x27b70a85, 0x2e1b2138, 0x4d2c6dfc, 0x53380d13,
            0x650a7354, 0x766a0abb, 0x81c2c92e, 0x92722c85,
            0xa2bfe8a1, 0xa81a664b, 0xc24b8b70, 0xc76c51a3,
            0xd192e819, 0xd6990624, 0xf40e3585, 0x106aa070,
            0x19a4c116, 0x1e376c08, 0x2748774c, 0x34b0bcb5,
```

```
                0x391c0cb3, 0x4ed8aa4a, 0x5b9cca4f, 0x682e6ff3,
                0x748f82ee, 0x78a5636f, 0x84c87814, 0x8cc70208,
                0x90befffa, 0xa4506ceb, 0xbef9a3f7, 0xc67178f2)
        #迭代初始值，h0,h1,...,h7
        self.h = h0

    #x 循环右移 b 个 bit
    #rightrotate b bit
    def rightrotate(self, x, b):
        return ((x >> b) | (x << (32 - b))) & ((2**32)-1)

    #信息预处理。附加填充和附加长度值
    def Pad(self, W):
        return bytes(W, "ascii") + b"\x80" + (b"\x00" * ((55 if (len(W) % 64) <
56 else 119) - (len(W) % 64))) + (
                (len(W) << 3).to_bytes(8, "big"))

    def Compress(self, Wt, Kt, A, B, C, D, E, F, G, H):
        return ((H + (self.rightrotate(E, 6) ^ self.rightrotate(E, 11) ^
self.rightrotate(E, 25)) + (
                (E & F) ^ (~E & G)) + Wt + Kt) + (
                                self.rightrotate(A, 2) ^ self.rightrotate(A, 13) ^
self.rightrotate(A, 22)) + (
                                        (A & B) ^ (A & C) ^ (B & C))) &
((2**32)-1), A, B, C, (D + (
                                        H + (self.rightrotate(E, 6) ^
self.rightrotate(E, 11) ^ self.rightrotate(E, 25)) + (
                                (E & F) ^ (~E & G)) + Wt + Kt)) & ((2**32)-1), E, F,
G

    def hash(self, message):
        message = self.Pad(message)
        digest = list(self.h)
        final=[]

        for i in range(0, len(message), 64):
            S = message[i: i + 64]
            W = [int.from_bytes(S[e: e + 4], "big") for e in range(0, 64, 4)] +
```

```
([0] * 48)

                #构造 64 个 word
                for j in range(16, 64):
                    W[j] = (W[j - 16] + (
                        self.rightrotate(W[j - 15], 7) ^ self.rightrotate(W[j - 15], 18) ^ (W[j - 15] >> 3)) + W[j - 7] + (self.rightrotate(W[j - 2], 17) ^ self.rightrotate(W[j - 2], 19) ^ (W[j - 2] >> 10))) & ((2**32)-1)

                A, B, C, D, E, F, G, H = digest

                for j in range(64):
                    A, B, C, D, E, F, G, H = self.Compress(W[j], self.constants[j], A, B, C, D, E, F, G, H)

            temp=(A, B, C, D, E, F, G, H)
            for t in range(8):
                final.append((digest[t]+temp[t])& ((2**32)-1))

            return final,"".join(format(h, "02x") for h in b"".join(
                d.to_bytes(4, "big") for d in [(x + y) & ((2**32)-1) for x, y in zip(digest, (A, B, C, D, E, F, G, H))]))



h0=(0x6a09e667, 0xbb67ae85, 0x3c6ef372, 0xa54ff53a,
    0x510e527f, 0x9b05688c, 0x1f83d9ab, 0x5be0cd19)

encoder = SHA256(h0)



message = "20190046"
mss_hash=encoder.hash(message)[0]
attack_mss="00440zyy"
length_attack=SHA256(mss_hash).hash(attack_mss)[1]
print(length_attack)
```

结论分析与体会：

对于正常的 sha256 算法而言，其初始的 h 值是固定的，会随着对每一个消息分块进行迭代后随之变化，这里需要做的就是对于 sha256 的实现进行一定的修改，使得其可以进入到任一轮次的迭代中。

先计算出前缀消息的哈希值，以 h 的形式（存有 8 个 32 位长数值的数组）输出，之后设置进行长度扩展攻击的后缀消息，将 h 和消息作为输入进行计算，可以得到长度扩展攻击的最终结果。