**一、SM3 哈希算法数学原理**

SM3 是中国国家密码管理局发布的加密哈希标准（GBT.32905-2016），在结构上类似于 SHA-256，但具有不同的压缩函数和参数。

**1. 消息填充（Padding）**

输入消息 M 被填充到 L ≡ 448 mod 512 位，并附加一个 64 位整数表示原始长度：

* 若消息长度为 l 位，填充方式为：

M || 1 || 00...0 || [l]\_64

**2. 分组处理**

将填充后的消息按 512 位划分为 n 个块：B0, B1, ..., Bn-1。

**3. 初始向量 IV**

IV = [0x7380166f, 0x4914b2b9, 0x172442d7, 0xda8a0600,

0xa96f30bc, 0x163138aa, 0xe38dee4d, 0xb0fb0e4e]

**4. 消息扩展（Message Expansion）**

将 512 位消息块扩展为 132 个 32 位字：

* W0...W67 基于线性变换 P1 和旋转
* W′j = Wj ⊕ Wj+4, j=0...63

**5. 压缩函数 CF**

对于每一个消息块，执行 64 轮迭代：

* 使用非线性函数 FFj, GGj：

FFj(X,Y,Z) = X ⊕ Y ⊕ Z (0 ≤ j < 16)

= (X & Y) | (X & Z) | (Y & Z) (16 ≤ j < 64)

GGj(X,Y,Z) = X ⊕ Y ⊕ Z (0 ≤ j < 16)

= (X & Y) | (~X & Z) (16 ≤ j < 64)

* 常数 Tj 不同于 SHA256 的使用方式：

text

复制编辑

Tj = 0x79cc4519 (0 ≤ j < 16), Tj = 0x7a879d8a (16 ≤ j < 64)

最终对每个消息块的输出与上一个中间结果异或后得到最终摘要。

**二、Length Extension Attack（长度扩展攻击）**

**1. 适用条件**

长度扩展攻击适用于 **基于 Merkle–Damgård 构造的哈希函数**（如 MD5、SHA1、SM3）。

前提条件：

* 已知 H(m)（不需要知道 m 本身）
* 希望伪造 H(m || padding || m') 的值

**2. 原理推导**

由于哈希函数是基于块压缩构造的，即：

复制编辑

H(m) = CF(CF(...CF(IV, B0), B1), ... Bn-1)

如果我们知道某个中间状态（如 H(m)），并知道 m 的长度，就可以从这个状态继续压缩新的数据 m'，构造出一个合法的哈希。

**3. 攻击步骤**

1. 知道 H(m) 和 len(m)
2. 构造填充后的 m || padding || m'
3. 设置初始状态为 H(m)，输入 m'，运行压缩函数
4. 得到 H(m || padding || m')，与原始哈希格式一致

**4. 防御方法**

* 使用 HMAC：将密钥作为哈希函数的一部分，攻击者无法模拟填充结构。
* 选择无长度扩展特性的哈希，如 SHA-3。

**三、RFC6962 Merkle 树数学原理（用于 CT 透明日志）**

**1. Merkle 树结构**

Merkle 树是一种二叉哈希树：

* 叶节点：Hash(0x00 || data)
* 中间节点：Hash(0x01 || left || right)

其中，0x00 和 0x01 是 RFC6962 中规定的 domain separation 前缀。

**2. Merkle Root 计算**

给定 n 个叶节点，计算方式：

build\_merkle\_tree(leaves):

if leaves.size() == 1:

return leaves[0]

split = largest\_power\_of\_two\_less\_than(n)

left\_root = build(leaves[0:split])

right\_root = build(leaves[split:n])

return Hash(0x01 || left\_root || right\_root)

**3. 叶子存在性证明（Inclusion Proof）**

**证明目标：某个叶子存在于某棵已知根的树中。**

证明结构：

* 给出 index i、叶子 d、proof [h1, h2, ..., hk]，这些是兄弟节点的哈希
* 使用递归组合：从 Hash(0x00 || d) 和 proof 构造 root
* 与原始 Merkle root 比较是否一致

验证流程：

hash = Hash(0x00 || leaf)

for (i = 0; i < proof.size(); ++i):

if (index % 2 == 0):

hash = Hash(0x01 || hash || proof[i])

else:

hash = Hash(0x01 || proof[i] || hash)

index = index / 2

**4. 不存在性证明（Non-Inclusion Proof）**

目标：证明某数据项不在树中

RFC6962 给出两种情况：

* 数据小于最左叶子或大于最右叶子：说明不在范围内
* 数据落在 [di, di+1) 区间中：提供这两个相邻叶子的存在性证明

核心思想：

若能证明目标落在两个**已知连续叶子之间**，且其哈希不同于二者，说明它不存在。

**实现思路部分**

**一、SM3 哈希函数实现思路**

SM3Hasher 类，复现了 SM3 国密哈希算法。主要模块如下：

**1. 压缩函数 compress()**

* 实现了消息扩展（W 和 W′）；
* 实现了 64 轮压缩循环；
* 使用 P0/P1、FF/GG、ROTL 等操作，还原了 SM3 的具体算法结构；
* 按照规范进行了状态的累加更新 V[i] ^= Ai。

**2. 消息填充 padding()**

* 实现标准的“附加1位+补0+附加64位长度”格式；
* 支持自定义 total\_len，为长度扩展攻击做准备；
* 返回填充后的完整消息块（64字节对齐）。

**3. 哈希函数 digest()**

* 接收任意输入；
* 对输入进行填充；
* 执行多个压缩函数后返回最终 256bit 哈希结果。

**4. 内部状态管理**

* hash\_to\_state()：将 32 字节摘要还原为 8 个 32 位整数状态；
* continue\_from\_state()：允许攻击者伪造一段追加数据，在已知哈希值前提下延续计算。

**二、长度扩展攻击实现思路**

复现典型的 length extension attack 场景，思路如下：

1. **攻击者已知：**
   * 原始消息 M 长度；
   * 哈希值 H(M)；
   * SM3 的填充机制。
2. **攻击步骤：**
   * 伪造消息 M || padding || append；
   * 取 H(M) 推出中间状态；
   * 构造填充好的 append 数据，继续压缩函数；
   * 得到伪造消息的哈希值 H(M || pad || append)。
3. **验证成功条件：**
   * 自己计算 H(M || pad || append) 与继续状态得到的 forged\_hash 一致；
   * 表示攻击者在不知道 M 的内容下伪造了合法 MAC。

**三、Merkle 树构建与验证思路（基于 RFC6962）**

设计一个支持 10 万叶子的 Merkle 树结构，核心思想如下：

**1. 叶子节点哈希**

leaf\_hash = Hash(0x00 || data);

* 前缀 0x00 表示叶节点；
* 保障叶子与内部节点哈希不冲突（遵循 RFC6962）。

**2. 内部节点哈希**

node\_hash = Hash(0x01 || left || right);

* 前缀 0x01 表示内部节点；
* 确保树结构安全。

**3. Merkle 树构建**

* 自底向上构造；
* 支持奇数个节点时“最后一个节点上浮”（即不复制）；
* 存储整个构建过程的每层节点便于后续证明。

**4. 存在性证明**

* 返回从叶子到根路径上的兄弟节点；
* 使用 verify\_inclusion() 方法验证路径哈希最终是否与根一致。

**5. 非存在性证明**

* 使用 lower\_bound 查找目标数据应在的位置；
* 返回其左右最近合法叶子的证明路径，作为“间隙”证据；
* 不直接证明“不存在”，而是通过“数据在已知区间之间”间接证明其未包含。

实验结果示例：

需要删除main函数注释后编译

