Project 4: SM3 的软件实现与优化
a）：与 Project 1 类似，从 SM3 的基本软件实现出发，参考付勇老师的 PPT，不断对 SM3 的软件执行效率进行改进
b）：基于 sm3 的实现，验证 length-extension attack
c）：基于 sm3 的实现，根据 RFC6962 构建 Merkle 树（10w 叶子节点），并构建叶子的存在性证明和不存在性证明

一. SM3 基础实现

代码结构

消息填充：附加比特"1" + 若干"0" + 64 位消息长度，确保总长度是 64 字节的整数倍

消息扩展：将 16 个 32 位字扩展为 68 个字（W）和 64 个字（W'）

前 16 字直接分块，后续字通过 P1 置换和循环移位生成：
W[j] = P1(W[j-16] ⊕ W[j-9] ⊕ rotl(W[j-3],15)) ⊕ rotl(W[j-13],7) ⊕ W[j-6]

轮函数处理：
每轮使用不同的布尔函数 FF/GG 和常量 T
寄存器更新：(A,B,C,D,E,F,G,H) ← (TT1,A,rotl(B,9),C,P0(TT2),E,rotl(F,19),G

最终输出：
将 8 个寄存器的值 A-H 与初始向量 IV 异或后拼接为 256 位哈希值。

二. SM3 简单优化

主要使用以下策略对其进行优化

计算层面优化
预计算常量：将 T_j 等常量预先计算存储，避免重复运算
循环展开：手动展开部分循环减少分支判断（如分 0-15 和 16-63 两阶段处理）
寄存器重用：通过变量重命名（如 D=C; C=rotl(B,9); B=A; A=TT1）减少内存访问

内存与 IO 优化
缓冲区管理：使用 bytearray 替代 bytes 避免频繁内存分配
批量处理：累积足够数据（≥64 字节）再触发压缩函数
零拷贝技术：通过 memoryview 直接操作内存缓冲区

指令级优化
位运算合并：将多个位移/异或操作合并（如 P0(x)=x⊕rotl(x,9)⊕rotl(x,17)）
局部变量缓存：将频繁访问的变量（如 self.V）缓存在局部变量中

优化效果

```
==== SM3 性能比较 ====
基础实现 平均耗时: 5.170306 秒
优化实现 平均耗时: 4.787826 秒
```

三．长度扩展攻击

```
==== Length-Extension 攻击 ====
Length-extension 攻击演示
原始消息哈希: 66c7f0f462eeedd9d1f2d46bdc10e4e24167c4875cf2f7a2297da02b8f4ba8e0
攻击者追加数据: b'def'
```

攻击本质
利用 Merkle-Damgård 结构的状态继承性：已知 H(m)和 len(m)，可计算 H(m‖pad‖suffix)而不需要知道 m。

攻击步骤
根据原始消息长度构造合法填充 pad
从 H(m)解析出内部状态寄存器值 A-H
以 A-H 为初始值继续计算 suffix 的哈希

四．Merkle 树构造

```
==== Merkle 树构建（10万叶子）性能测试 ====
构建10万叶子节点的Merkle树耗时: 101.569 秒
```

构建步骤
准备数据：将 10 万个数据项作为叶子节点
计算叶子哈希：对每个数据项计算 SM3 哈希值
构建树结构：
第一层：10 万个叶子节点
第二层：每两个相邻叶子节点哈希拼接后计算 SM3 哈希，得到约 5 万个节点

重复此过程直到只剩一个根节点

数学表示
对于节点 N，其哈希计算为：
如果是叶子节点：H(0x00 || data)
如果是内部节点：H(0x01 || left_child_hash || right_child_hash)
如果节点数为奇数，最后一个节点直接升级

五．Merkle 树节点存在性和不存在性证明

```
==== Merkle 树测试（小规模） ====
Merkle 根: 7d766fb3615850c699bbd9bcab320a19b9bced76a0fe9b768f2fd23e138d655d
叶子3的包含证明: ['caddd6ce69a54bd2945d0dd21519dbdd6bd0b7eb30619e82604b9e30b998b2069', '1e1e5afe77b0c89cfc073050630f49c988f3967a30725e066b3
非包含证明: 非包含证明
```

存在性证明(Proof of Inclusion)
概念
证明某个特定数据项确实存在于 Merkle 树中。

证明构成
目标叶子节点的哈希值
从该叶子到根的路径上所有兄弟节点的哈希值(称为"认证路径")
通过这些哈希值可以重新计算出 Merkle 根，与已知根匹配则证明成立

数学过程
给定叶子节点 L，其认证路径为[h1, h2, ..., hn]：
计算 current_hash = H(L)
对于每个 hi 在路径中：
如果 hi 是左兄弟：current_hash = H(hi || current_hash)
如果 hi 是右兄弟：current_hash = H(current_hash || hi)
最终 current_hash 应与 Merkle 根一致

不存在性证明(Proof of Exclusion)
概念
证明某个特定数据项不存在于 Merkle 树中。

证明构成
两个相邻叶子节点的存在性证明，它们按排序应包含目标值
证明这两个叶子节点确实是相邻的

数学过程
假设树中数据按字典序排序，要证明 X 不存在：
找到树中最大的小于 X 的叶子节点 L 和最小的大于 X 的叶子节点 R
提供 L 和 R 的存在性证明
证明 L 和 R 在树中确实是相邻节点(通过展示它们共享某个祖先节点)