说明文档

1 实验内容与实现细节

1.1 SM3 基础实现与优化

基础实现

在 `sm3.hpp` / `sm3.cpp` 中实现了 SM3 算法的核心逻辑，包括：

- 状态变量 `digest_` 的初始化
- 数据分块与缓冲区管理
- 消息扩展与压缩函数
- 布尔函数 FF / GG 与置换函数 P0 / P1

接口包括：

```
void update(const uint8_t* data, size_t len);
void final(uint8_t out[HASH_SIZE]);
void reset();
```

优化措施

1. **循环展开与位运算优化**\
   将部分轮函数展开，减少循环索引计算开销。
2. **缓冲区管理改进**\
   在 `update()` 中减少多余的 memcpy 调用，批量处理分组。

1.2 长度扩展攻击验证

利用 `set_state()` 接口直接设置中间状态与已处理的比特数，模拟攻击者在不知道原始消息的情况下，追加数据并计算合法哈希。

流程

1. 已知原始消息 `m` 及其哈希 `H(m)`。
2. 根据 SM3 初始值或已知状态，调用 `set_state()`。
3. 向 SM3 实例输入追加数据 `m2`，得到新的哈希 `H(m || padding || m2)`。

代码示例

```
std::array<uint32_t,8> state = {...}; // 已知的中间状态
sm3_le.set_state(state, msg.size() * 8);
sm3_le.update((const uint8_t*)m2.c_str(), m2.size());
sm3_le.final(le_hash);
```

1.3 Merkle 树实现

叶子哈希生成

每个叶子节点的哈希由原始数据通过 SM3 计算得到。

树的构建

- 采用完全二叉树构造方法
- 每个父节点的哈希由其左右子节点哈希拼接后再计算 SM3

存在性证明

- 给定叶子节点索引，依次收集从叶子到根路径上的兄弟节点哈希
- `verify_existence()` 根据叶子哈希与路径重算根哈希，验证一致性

不存在性证明

- 找到目标索引左、右两侧最近存在的叶子
- 分别生成这两个叶子的存在性证明
- 若两者在顺序上正好包围目标位置，则证明目标索引不存在

2 实验结果

```
SM3("hello sm3") = ce2512f4a1487ff23eab376950a7d525cba630696abababadc1136531372e6cce3
Length-extension hash = 7fb4bd94a0e5bf42321918ed57544c8387bbbd2ca2c33a5147821b521e3b8fd4
Building Merkle tree...
Merkle root hash: 5b1a12848c0f80ce0147051f5ccc75ecbcf84dc7b04c93b4947410a26b562fdb
Existence verification for leaf 12345 : FAIL
Non-existence verification : FAIL
```