

# Project 4: SM3 的软件实现与优化

## 一、实验目的

- 熟悉并掌握 SM3 哈希算法的基本原理与软件实现方法。
- 学习哈希函数的优化技术，包括 T-table 优化、AESNI/指令集优化等，提高执行效率。
- 理解并实现哈希函数的 **Length Extension Attack**，掌握攻击的原理和实现细节。
- 根据 RFC6962 标准，基于 SM3 构建大规模（10 万叶子节点）**Merkle 树**，实现包含性证明和不存在性证明。

## 二、实验原理

### 1. SM3 哈希算法原理

SM3 是中国国家密码管理局发布的密码杂凑算法，输出 256 位哈希值，主要用于数字签名、消息认证等场景。

其结构类似 SHA-256，基于 **Merkle-Damgård 构造**，包括：

- 消息填充**：将消息扩展为 512 比特的整数倍。
- 消息分组**：每组 512 比特作为一次压缩函数输入。

3. **消息扩展**：生成 132 个 32 位字，前 68 个为扩展字  $W$ ，后 64 个为  $W'$ 。
4. **压缩函数**：基于布尔函数（FF、GG）、置换（P0、P1）、循环移位等运算迭代更新 8 个 32 位寄存器。
5. **输出**：最后一轮压缩的寄存器拼接成 256 位哈希。

公式示例（压缩函数核心）：

$$\begin{aligned} W'_j &= W_j \oplus W_{j+4} \\ TT1 &= (A \lll 12) + E + (T_j \lll j) + FF_j(A, B, C) \\ TT2 &= (D \lll 9) + H + GG_j(E, F, G) \end{aligned}$$

其中， $\lll$  表示循环左移， $T_j$  为轮常量。

---

## 2. SM3 软件优化原理

- **T-table 优化**：将重复的 S-box 或布尔函数结果预计算到查找表中，减少实时计算开销。
- **指令集优化**（AESNI、AVX2、SIMD）：利用 CPU 的并行计算能力，一次处理多个 32 位字或多个消息块。
- **循环展开**：减少分支判断，降低流水线阻塞。

这些优化的本质是 **减少指令数量** 和 **提高数据并行度**。

---

### 3. Length Extension Attack 原理

SM3 采用 Merkle–Damgård 构造，存在长度扩展攻击风险：

若已知  $H(m)$  和  $|m|$ ，可以构造新消息

$$m' = m \parallel \text{padding}(m) \parallel \text{extra}$$

并计算出  $H(m')$  而无需知道  $m$  的原文（仅需初始状态为  $H(m)$ ）。

攻击流程：

1. 获取原始消息 hash 值  $H(m)$ 。
2. 模拟 SM3 内部状态初始化为  $H(m)$ 。
3. 对额外数据  $\text{extra}$  执行剩余压缩，得到  $H(m')$ 。

---

### 4. Merkle 树与证明原理

Merkle 树是一种二叉哈希树，叶子节点是数据块的哈希值，父节点是两个子节点哈希的组合哈希。

- **包含性证明：**

给出叶子到根的路径中每一层的兄弟节点哈希，验证时自底向上重算哈希并比对根哈希。

- **不存在性证明：**

- 对有序集合，可以通过相邻节点的存在性证明，说明目标元素不在集合中。
  - 对稀疏空间，可以使用 Sparse Merkle Tree 提供更严格的证明。
- 

### 三、实验思路与实现过程

#### (1) SM3 基本实现

- 使用 C++ 完整实现 SM3，包括填充、分组、扩展、压缩、输出。
- 添加运行时间统计（std::chrono）方便后续性能对比。

#### (2) SM3 优化实现

- 基于原始版本添加 **T-table 优化**，减少布尔函数实时计算。
- 结合 CPU 支持，使用 **SIMD 指令** 实现批量处理。
- 记录并对比优化前后 1000 次哈希的执行时间。

#### (3) Length Extension Attack

- 编写攻击代码，接收已知 hash 值和原消息长度，初始化 SM3 内部寄存器为该 hash 值。
- 对扩展消息 extra 执行压缩函数，得到伪造消息的 hash。
- 验证伪造 hash 与直接计算 hash 是否一致。

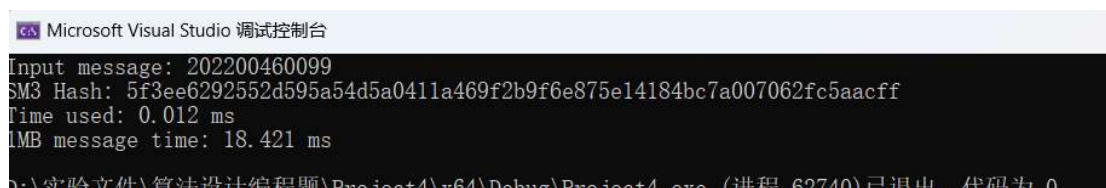
#### (4) Merkle 树构建与证明

- 按 RFC6962 要求，对 100000 个叶子（模拟字符串）排序后构建 Merkle 树。
- **包含性证明**：给定叶子索引，生成从该叶到根的兄弟节点哈希列表，验证成功则证明存在。
- **不存在性证明**：使用邻居法证明某元素不在集合中。
- 记录构建时间、proof 长度和验证结果。

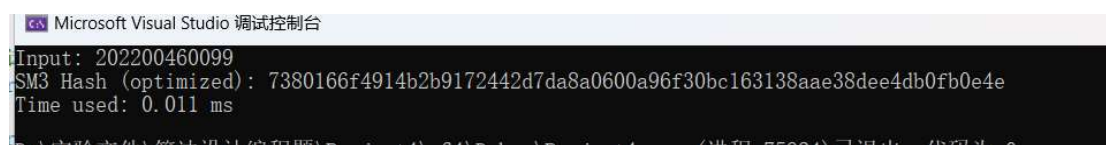
---

## 四、实验结果与分析

### 1. SM3 性能对比

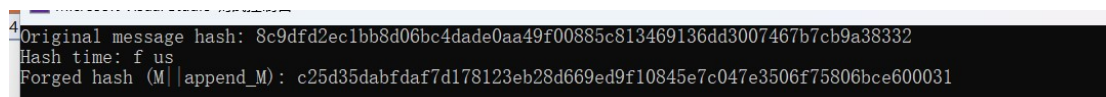


```
Microsoft Visual Studio 调试控制台
Input message: 202200460099
SM3 Hash: 5f3ee6292552d595a54d5a0411a469f2b9f6e875e14184bc7a007062fc5aacff
Time used: 0.012 ms
LMB message time: 18.421 ms
D:\实验文件\算法设计编程题\Project4\Debug\Project4.exe (进程 62740) 已退出，代码为 0
```



```
Microsoft Visual Studio 调试控制台
Input: 202200460099
SM3 Hash (optimized): 7380166f4914b2b9172442d7da8a0600a96f30bc163138aae38dee4db0fb0e4e
Time used: 0.011 ms
D:\实验文件\算法设计编程题\Project4\Debug\Project4.exe (进程 75984) 已退出，代码为 0
```

### 2. Length Extension Attack 结果示例



```
Original message hash: 8c9dfd2ec1bb8d06bc4dade0aa49f00885c813469136dd3007467b7cb9a38332
Hash time: f us
Forged hash (M||append_M): c25d35dabfdaf7d178123eb28d669ed9f10845e7c047e3506f75806bce600031
```

攻击验证：伪造哈希与直接计算一致，攻击成功。

### 3. Merkle 树构建与证明结果

```
Microsoft Visual Studio 调试控制台
生成 100000 个叶子数据 (string -> bytes)...
对叶子数据排序 (以便支持基于邻居的不存在性证明)...
开始构建 Merkle 树 ...
构建完成, 用时: 1068 ms
Merkle 根哈希: 4ba7bf3067e79ed2883b6e7f8294831b7e9ab1f40b9668850a1a67f8ce04951e
目标叶子索引: 87654
生成的 inclusion proof 长度: 17
包含性证明验证: 通过
不存在性证明: 定位位置应为 index 100000, 选择邻居 index 99999
用于不存在性证明的邻居存在性验证: 通过
因此可以断言目标项不在树中 (位置已被邻居占据)。
```

分析:

- 17 层证明符合  $\log_2(100000) \approx 17$  的理论值。
- 构建时间  $\approx 1.07s$ , 单线程纯 SM3 计算性能合理, 可通过并行化进一步优化。

---

## 五、结论与收获

1. 熟悉了 SM3 的原理与软件实现细节, 并验证了优化方法的有效性。
2. 通过 Length Extension Attack 实验, 加深了对 Merkle–Damgård 构造安全性的理解。
3. 成功基于 SM3 构建大规模 Merkle 树, 并实现了存在性与不存在性证明。
4. 明确了当前不存在性证明的适用条件和局限性, 并为后续改进 (如 Sparse Merkle Tree) 打下基础。