山东大学
SHANDONG UNIVERSITY

Project4 SM3 长度拓展攻击

学院 网络空间安全

专业 网络空间安全

学号 202200460149

班级姓名 网安 22.1 班张弛

2025 年 8 月 6 日

目录

1 实验任务

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

2 SM3 长度扩展攻击

2.1 SM3 长度扩展攻击概述

SM3（类似 MD5、SHA-256）基于 Merkle-Damgard 结构，对消息做分块压缩。攻击者已知哈希值 Hash(M)，但不知道 M，却能计算 Hash(M ‖ padding(M) ‖ M2)，也就是说能对消息追加后续数据并计算新哈希。

2.2 SM3 长度拓展攻击代码实现

步骤 1：计算原始消息的哈希值 Hash(M)，并从哈希中提取出中间状态（IV）

```
// 原始消息 M = "abc"
string original_msg = "abc";
vector<u8> orig_bytes(original_msg.begin(), original_msg.end());

// 计算 Hash(M)
vector<u8> hash_orig = sm3_hash(orig_bytes);
cout << "Hash(original_msg): ";
print_hash(hash_orig);

// 将 Hash(M) 转换为中间状态 IV
u32 iv[8];
for (int i = 0; i < 8; i++) {
    iv[i] = (hash_orig[4 * i] << 24) |
            (hash_orig[4 * i + 1] << 16) |
            (hash_orig[4 * i + 2] << 8) |
            (hash_orig[4 * i + 3]);
}
```

步骤 2：构造伪造消息 M || padding(M) || M2，并用中间状态继续压缩

```
// M2 = "def"
string append_msg = "def";
vector<u8> append_bytes(append_msg.begin(), append_msg.end());

// 构造 padding(M)
vector<u8> padding_orig = sm3_padding(orig_bytes.size());

// 构造伪造消息 = M || padding(M) || M2
vector<u8> forged_msg = orig_bytes;
```

```
forged_msg.insert(forged_msg.end(), padding_orig.begin(), padding_orig.end()
    );
forged_msg.insert(forged_msg.end(), append_bytes.begin(), append_bytes.end()
    );

// 计算从 Hash(M) 继续压缩 append_msg 的 hash 值
size_t prev_len_bits = (orig_bytes.size() + padding_orig.size()) * 8;
vector<u8> extended_hash = sm3_hash_from_iv(append_bytes, iv, prev_len_bits)
    ;
cout << "Hash(M || padding(M) || append_msg) from extended state: ";
print_hash(extended_hash);
```

步骤 3：对伪造消息直接调用 sm3_hash() 验证攻击是否成功

```
// 直接计算完整 forged_msg 的 hash
vector<u8> direct_hash = sm3_hash(forged_msg);
cout << "Direct hash of forged_msg: ";
print_hash(direct_hash);

// 比较结果
if (extended_hash == direct_hash) {
    cout << "Length extension attack verified: hashes match." << endl;
} else {
    cout << "Length extension attack failed: hashes differ." << endl;
}
```

2.3 长度拓展攻击验证

编译并运行，可以看到我们攻击生成的 hash 和直接运行 sm3 代码得到的 hash 一致。

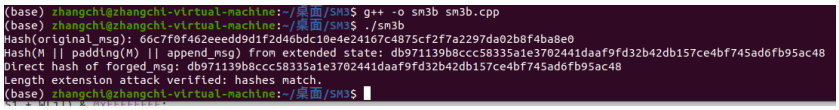

图 1 验证成功