1. SM3 算法概述

SM3 是中国国家密码管理局于2010年发布的密码杂凑算法标准，输出长度为256位（32字节）。它适用于数字签名、消息认证码生成与验证、随机数生成等密码应用。

1.1 算法特点

• 抗碰撞性：难以找到两个不同的输入产生相同的哈希值

• 单向性：难以从哈希值反推原始输入

• 雪崩效应：输入微小变化导致输出巨大变化

2. 算法流程

2.1 消息填充

将消息填充为512位（64字节）的倍数：

1. 在消息末尾添加一个'1'比特（即字节0x80）

2. 添加k个'0'比特，使得填充后长度模512等于448

3. 最后64位表示原始消息长度的比特数（大端序）

数学表示：

设原始消息长度为 l 比特，填充后的消息为：

M' = M \parallel 1 \parallel 0^k \parallel (l)\_{64}

其中 k 是满足 (l + 1 + k) \equiv 448 \pmod{512} 的最小非负整数

2.2 消息扩展

将512位的消息分组划分为132个32位字（W\_0到W\_{67}和W'\_0到W'\_{63}）：

1. 将分组划分为16个32位字 W\_0, W\_1, \ldots, W\_{15}

2. 对于 j 从16到67：

W\_j = P\_1(W\_{j-16} \oplus W\_{j-9} \oplus (W\_{j-3} \lll 15)) \oplus (W\_{j-13} \lll 7) \oplus W\_{j-6}

3. 对于 j 从0到63：

W'\_j = W\_j \oplus W\_{j+4}

其中 P\_1(X) = X \oplus (X \lll 15) \oplus (X \lll 23)

2.3 压缩函数

压缩函数使用256位的状态（8个32位寄存器A,B,C,D,E,F,G,H）和消息分组进行迭代：

1. 初始化寄存器为固定初始值（IV）

2. 对于每个分组，执行64轮迭代

3. 每轮使用不同的布尔函数和常量：

• 布尔函数 FF\_j 和 GG\_j（0 \leq j < 64）：

FF\_j(X,Y,Z) =

\begin{cases}

X \oplus Y \oplus Z & 0 \leq j < 16 \\

(X \land Y) \lor (X \land Z) \lor (Y \land Z) & 16 \leq j < 64

\end{cases}

GG\_j(X,Y,Z) =

\begin{cases}

X \oplus Y \oplus Z & 0 \leq j < 16 \\

(X \land Y) \lor (\neg X \land Z) & 16 \leq j < 64

\end{cases}

• 常量 T\_j：

T\_j =

\begin{cases}

0x79cc4519 & 0 \leq j < 16 \\

0x7a879d8a & 16 \leq j < 64

\end{cases}

4. 每轮计算：

SS1 = ((A \lll 12) + E + (T\_j \lll j)) \lll 7

SS2 = SS1 \oplus (A \lll 12)

TT1 = FF\_j(A,B,C) + D + SS2 + W'\_j

TT2 = GG\_j(E,F,G) + H + SS1 + W\_j

D = C

C = B \lll 9

B = A

A = TT1

H = G

G = F \lll 19

F = E

E = P\_0(TT2)

其中 P\_0(X) = X \oplus (X \lll 9) \oplus (X \lll 17)

5. 一轮结束后，更新状态：

(A,B,C,D,E,F,G,H) \leftarrow (A \oplus A\_0, B \oplus B\_0, \ldots, H \oplus H\_0)

其中右边是初始状态（上一分组结束后的状态）和本轮计算得到的新状态的异或

2.4 输出

处理完所有分组后，将最后的状态寄存器A,B,C,D,E,F,G,H连接起来，得到256位的杂凑值

3. 长度扩展攻击

3.1 攻击原理

长度扩展攻击针对基于Merkle-Damgård结构的哈希函数（如SM3）。攻击者知道H(m)和m的长度（但不知道m），可以计算H(m \parallel pad \parallel m')，其中pad是m的填充。

3.2 攻击步骤

1. 已知H(m)和len(m)

2. 计算m的填充pad，使得m \parallel pad的长度是512位的倍数

3. 构造新消息m'' = m \parallel pad \parallel m'

4. 将H(m)作为初始状态，计算H(m'')，其中处理的消息块是pad和m'的填充消息

3.3 数学表示

设原始消息 m 的哈希为 H(m)，长度为 l，攻击者构造新消息：

m'' = m \parallel pad(l) \parallel m'

则：

H(m'') = \text{SM3}(H(m), pad(l) \parallel m' \parallel \text{填充}(pad(l) \parallel m'))

3.4 防御措施

• 使用HMAC结构：HMAC(K, m) = H((K \oplus opad) \parallel H((K \oplus ipad) \parallel m))

• 使用其他抗长度扩展攻击的哈希结构（如SHA-3）

1. Merkle树（RFC6962）

4.1 基本概念

Merkle树（哈希树）是一种二叉树结构，用于高效验证大量数据的完整性。RFC6962定义了用于证书透明化的Merkle树结构。

4.2 构建过程

• 叶子节点：对数据块D\_i计算哈希H(0x00 \parallel D\_i)

• 内部节点：对两个子节点L和R计算H(0x01 \parallel L \parallel R)

• 如果节点数为奇数，复制最后一个节点

数学表示：

设叶子节点集合为 L = \{L\_0, L\_1, \ldots, L\_{n-1}\}，则：

\text{叶子哈希} = H(0x00 \parallel D\_i)

\text{父节点哈希} = H(0x01 \parallel \text{左子节点} \parallel \text{右子节点})

4.3 存在性证明

对于叶子节点D\_i，提供从该叶子节点到根节点的路径上的所有兄弟节点。验证时：

1. 计算叶子哈希 h\_0 = H(0x00 \parallel D\_i)

2. 对于路径上的每个兄弟节点 (s\_i, \text{is\\_right})：

• 如果 \text{is\\_right} 为真：h\_{i+1} = H(0x01 \parallel h\_i \parallel s\_i)

• 否则：h\_{i+1} = H(0x01 \parallel s\_i \parallel h\_i)

3. 最终哈希 h\_n 应与根哈希一致

4.4 不存在性证明

证明一个数据块D不存在于树中：

1. 找到两个叶子节点D\_a和D\_b，使得D\_a < D < D\_b（按字典序）

2. 证明D\_a和D\_b在树中是相邻的

3. 提供D\_a和D\_b的存在性证明

数学验证：

1. 验证 D\_a 和 D\_b 的存在性

2. 验证 D\_a 和 D\_b 相邻：它们之间没有其他叶子节点

3. 验证 D\_a < D < D\_b，证明 D 不存在于树中

4.5 RFC6962 特定规定

• 哈希函数：使用抗碰撞的哈希函数（如SM3）

• 叶子节点前缀：0x00

• 内部节点前缀：0x01

• 序列化：所有数据按大端序处理

5. 优化策略

5.1 软件优化技术

1. 循环展开：减少分支预测开销

• 将主循环展开为每次处理4轮

2. SIMD指令：使用AVX2等指令集并行处理多个消息块

3. 查表优化：预计算常量表，避免重复计算

4. 内存对齐：确保数据结构对齐，提高内存访问效率

5. 内联函数：减少函数调用开销