

SM3 密码杂凑算法

王小云^{1,2} 于红波³

¹(清华大学高等研究院 北京 100084)

²(密码技术与信息安全教育部重点实验室(山东大学) 济南 250100)

³(清华大学计算机系 北京 100084)

(xiaoyunwang@mail.tsinghua.edu.cn)

SM3 Cryptographic Hash Algorithm

Wang Xiaoyun^{1,2} and Yu Hongbo³

¹(Institute for Advanced Study, Tsinghua University, Beijing 100084)

²(Key Laboratory of Cryptologic Technology and Information Security (Shandong University), Ministry of Education, Jinan 250100)

³(Department of Computer Science and Technology, Tsinghua University, Beijing 100084)

Abstract The cryptographic hash functions play an important role in modern cryptography. They are used to compress messages of arbitrary length to fixed length hash values. The most common cryptographic applications of hash functions are with digital signature and for data integrity. SM3 cryptographic hash algorithm is issued as the industry standard in 2012. In 2016, it was published as national standard. It takes a 512-bit message as input and outputs a 256-bit hash value. This paper summarizes the design, properties, software and hardware implementations and cryptanalysis of SM3 cryptographic hash algorithm. Furthermore, we compare SM3 with other hash standards.

Key words SM3 algorithm; cryptographic hash function; collision attack; preimage attack; distinguishing attack

摘要 密码杂凑算法是3类基础密码算法之一,它可以将任意长度的消息压缩成固定长度的摘要,主要用于数字签名和数据完整性保护等。SM3密码杂凑算法的消息分组长度为512 b,输出摘要长度为256 b。该算法于2012年发布为密码行业标准(GM/T 0004—2012),2016年发布为国家密码杂凑算法标准(GB/T 32905—2016)。总结了SM3密码杂凑算法的设计原理、算法特点、软硬件实现和安全性分析,同时将SM3密码杂凑算法和国际通用杂凑算法在实现效率和安全性方面进行比较。

关键词 SM3 算法;密码杂凑算法;碰撞攻击;原像攻击;区分攻击

中图法分类号 TP309

收稿日期:2016-10-25

基金项目:国家“九七三”重点基础研究发展规划项目(2013CBB34200);国家自然科学基金项目(61133013);清华信息科学与技术国家实验室基金项目

密码杂凑算法在现代密码学中起着重要作用,它可以将任意长度的消息压缩成固定长度的摘要。杂凑算法是密码学的3类基础算法之一(加密算法、数字签名算法和杂凑算法),主要用于数据的完整性校验、身份认证、数字签名、密钥推导、消息认证码和随机比特生成器等。

密码杂凑算法需要满足3个基本属性:抗碰撞攻击、抗原像攻击和抗第二原像攻击。随着杂凑算法分析技术的进步,杂凑算法的安全属性不再局限于3个基本属性,还出现了许多其他属性,比如抗长度扩展攻击、抗长消息的第二原像攻击^[1]和抗集群攻击^[2]等。

2004—2005年我国密码学家王小云等人破解了国际通用系列杂凑算法,包括MD5^[3],SHA-1^[4],RIPEMD^[3],HAVAL^[3]等,引起国际密码社会的强烈反响。为了应对MD5与SHA-1的破解^[3-5],NIST于2007—2012年开展了公开征集新一代杂凑算法标准SHA-3^[6]。SHA-3竞赛征集到了64个算法,这些候选算法各具特色,体现了很多新的设计理念。进入SHA-3最终轮的5个候选算法都采用了不同于MD结构的新型结构:KECCAK采用海绵体结构,BLAKE采用HAIFA结构,Skein采用基于密文的唯一分组迭代(unique block iteration)链接模式,Grøstl和JH是基于宽管道的MD改进结构。在内部变换中,KECCAK采用基于3维数组的比特级逻辑运算;BLAKE和Skein基于加、循环移位和异或(ARX)运算;Grøstl采用AES类的设计;JH使用了扩展的多维AES结构。经过5年的遴选,KECCAK凭借其优美的设计、足量的安全冗余、出色的整体表现、高效的硬件效率和适当的灵活性最终胜出成为SHA-3标准。

随着SHA-3竞赛的进行,各个国家都在设计相应的杂凑算法标准。2012年,国家商用密码管理办公室公布了SM3密码杂凑算法^[7]为密码行业标准。2016年,国家标准化委员会公布了SM3密码杂凑算法为国家标准^[8]。SM3密码杂凑算法已经提交ISO国际标准化组织,目前已进入DIS阶段。本文对SM3密码杂凑算法的设计原理、算法特点、软硬件实现和安全性分析进行介绍,同时给出了SM3密码杂凑算法与国内外其他算法的比较结果。

1 SM3密码杂凑算法描述

SM3密码杂凑算法采用Merkle-Damgård结构,消息分组长度为512 b,摘要长度256 b。压缩函数状态256 b,共64步操作。本节给出了SM3密码杂凑算法的描述和特点。

1.1 SM3密码杂凑算法的描述

1.1.1 SM3密码杂凑算法的初始值

SM3密码杂凑算法的初始值IV共256 b,由8个32 b串联构成,具体值如下:

$$\begin{aligned} IV = & 7380166f\ 4914b2b9\ 172442d7\ da8a0600 \\ & a96f30bc\ 163138aa\ e38dee4d\ b0fb0e4e. \end{aligned}$$

1.1.2 SM3密码杂凑算法的常量

SM3密码杂凑算法的常量 T_j 定义如下:

$$T_j = \begin{cases} 79cc4519, & 0 \leq j \leq 15, \\ 7a879d8a, & 16 \leq j \leq 63. \end{cases}$$

1.1.3 SM3密码杂凑算法的布尔函数

SM3密码杂凑算法的布尔函数定义如下:

$$FF_j(X, Y, Z) =$$

$$\begin{cases} X \oplus Y \oplus Z, & 0 \leq j \leq 15, \\ (X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z), & 16 \leq j \leq 63, \end{cases}$$

$$GG_j(X, Y, Z) =$$

$$\begin{cases} X \oplus Y \oplus Z, & 0 \leq j \leq 15, \\ (X \wedge Y) \vee (\neg X \wedge Z), & 16 \leq j \leq 63. \end{cases}$$

1.1.4 SM3密码杂凑算法的置换函数

SM3密码杂凑算法的置换函数定义如下:

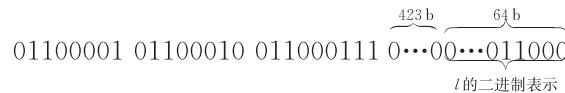
$$P_0(X) = X \oplus (X \lll 9) \oplus (X \lll 17),$$

$$P_1(X) = X \oplus (X \lll 15) \oplus (X \lll 23).$$

1.1.5 SM3密码杂凑算法的消息填充

对长度为 $l(l < 2^{64})$ 比特的消息 m ,SM3密码杂凑算法首先将比特“1”添加到消息的末尾,再添加 k 个“0”, k 是满足 $l+k+1 \equiv 448 \pmod{512}$ 的最小非负整数。然后再添加一个64位比特串,该比特串是长度 l 的二进制表示。填充后的消息 m' 的比特长度为512的倍数。

例如:对消息01100001 01100010 01100011,其长度 $l=24$,经填充得到的比特串如下:



1.1.6 SM3 密码杂凑算法的迭代压缩过程

将填充后的消息 m' 按 512 b 进行分组: $m' = B^{(0)}B^{(1)}\cdots B^{(n-1)}$, 其中 $n = (l+k+65)/512$. 对 m' 按如下方式迭代:

```
FOR i=0 TO (n-1)
     $V^{(i+1)} = CF(V^{(i)}, B^{(i)})$ ;
ENDFOR
```

其中, CF 是压缩函数, $V^{(0)}$ 为 256 b 初始值 IV , $B^{(i)}$ 为填充后的消息分组, 迭代压缩的结果为 $V^{(n)}$.

1.1.7 SM3 密码杂凑算法的压缩函数

SM3 密码杂凑算法的压缩函数由消息扩展过程和状态更新过程组成, 具体描述如下.

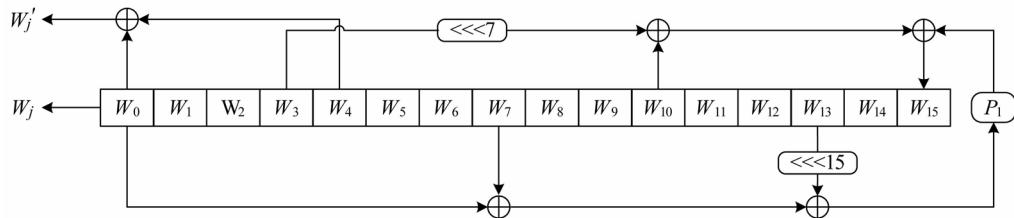


图 1 SM3 消息扩展过程

过程 2. 状态更新过程.

假定 A, B, C, D, E, F, G, H 为寄存器, $SS1, SS2, TT1, TT2$ 为中间变量, 压缩函数 $V^{(i+1)} = CF(V^{(i)}, B^{(i)})$, $0 \leq i \leq n-1$, 状态更新过程描述如下(如图 2 所示):

```
ABCDEFGH  $\leftarrow V^{(i)}$ ;
FOR j=0 TO 63
    SS1  $\leftarrow ((A \lll 12) + E + (T_j \lll j))$ 
     $\lll 7$ ;
    SS2  $\leftarrow SS1 + (A \lll 12)$ ;
    TT1  $\leftarrow FF_j(A, B, C) + D + SS2 + W'_j$ ;
```

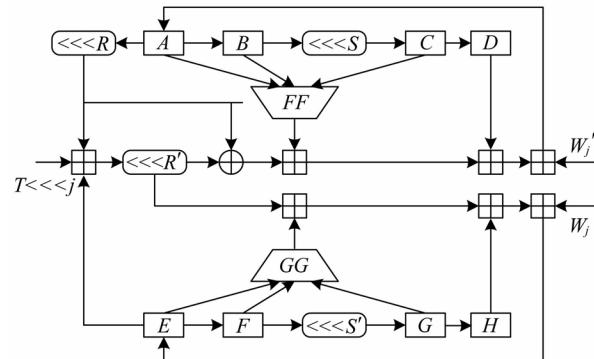


图 2 SM3 状态更新过程

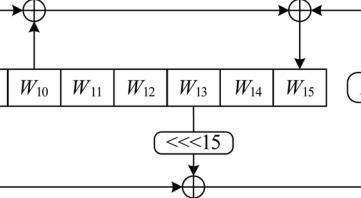
过程 1. 消息扩展过程.

将消息分组 $B^{(i)}$ 按以下方式扩展生成 132 个字 $W_0, W_1, \dots, W_{67}, W'_0, W'_1, \dots, W'_{63}$ 用于压缩函数 CF (如图 1 所示):

- 1) 将消息分组 $B^{(i)}$ 划分为 16 个字 W_0, W_1, \dots, W_{15} ;
- 2) FOR $j=16$ TO 67

$$W_j = P_1(W_{j-16} \oplus W_{j-9} \oplus (W_{j-3} \lll 15) \oplus (W_{j-1} \lll 7) \oplus W_{j-6})$$
- ENDFOR
- 3) FOR $j=0$ TO 63

$$W'_j = W_j \oplus W_{j+4}$$
- ENDFOR



$TT2 \leftarrow GG_j(A, B, C) + H + SS1 + W_j$;

$D \leftarrow C$;

$C \leftarrow B \lll 9$;

$B \leftarrow A$;

$A \leftarrow TT1$;

$H \leftarrow G$;

$G \leftarrow F \lll 19$;

$F \leftarrow E$;

$E \leftarrow P_0(TT2)$;

ENDFOR

$V^{(i+1)} \leftarrow ABCDEFGH \oplus V^{(i)}$.

过程 3. 杂凑值.

$ABCDEFGH \leftarrow \oplus V^{(n)}$

输出 256 b 的杂凑值 $y = ABCDEFGH$.

1.2 SM3 密码杂凑算法的特点

SM3 密码杂凑算法压缩函数整体结构与 SHA-256 相似,但是增加了多种新的设计技术,包括增加 16 步全异或操作、消息双字介入、增加快速雪崩效应的 P 置换等. 能够有效地避免高概率的局部碰撞,有效地抵抗强碰撞性的差分分析、弱碰撞性的线性分析和比特追踪法等密码分析.

SM3 密码杂凑算法合理使用字加运算,构成进位加 4 级流水,在不显著增加硬件开销的情况下,采用 P 置换,加速了算法的雪崩效应,提高了运算效率。同时,SM3 密码杂凑算法采用了适合 32 b 微处理器和 8 b 智能卡实现的基本运算,具有跨平台实现的高效性和广泛的适用性。

2 SM3 密码杂凑算法的设计原理

SM3 密码杂凑算法的设计主要遵循以下原则:1) 能够有效抵抗比特追踪法及其他分析方法;2) 软硬件实现需求合理;3) 在保障安全性的前提下,综合性能指标与 SHA-256 同等条件下相当。

2.1 压缩函数的设计

2.1.1 设计原则

压缩函数的设计具有结构清晰、雪崩效应强等特点,采用了以下设计技术:

- 1) 消息双字介入。输入的双字消息由消息扩展算法产生的消息字中选出。为了使介入的消息尽快产生雪崩效应,采用了模 2^{32} 算术加运算和 P 置换等。

- 2) 每一步操作将上一步介入的消息比特非线性迅速扩散,每一消息比特快速地参与进一步的扩散和混乱。

- 3) 采用混合来自不同群运算,模 2^{32} 算术加运算、异或运算、3 元布尔函数和 P 置换。

- 4) 在保证算法安全性的前提下,为兼顾算法的简介和软硬件及智能卡实现的有效性,非线性运算主要采用布尔运算和算术加运算。

- 5) 压缩函数参数的选取应使压缩函数满足扩散的完全性、雪崩速度快的特点。

2.1.2 P_0 置换的参数选取

P_0 置换参数选取需要排除位移间距较短、位移数为字节倍数和位移数都为合数的情况,综合考虑算法设计的安全性、软件和智能卡实现的效率,选取移位常量为 9 和 17。

2.1.3 布尔函数的选取

布尔函数的作用主要是用于防止比特追踪法、提高算法的非线性特性和减少差分特征的遗传等。因此,布尔函数的选取需要满足以下要求:

- 1) 0~15 步布尔函数采用全异或运算,以防止比特追踪法分析。

- 2) 16~63 步布尔函数采用非线性运算,提高算法的非线性特性。同时,需要满足差分分布均匀,与压缩函数中的移位运算结合,以减少输入和输出间的差分特征遗传。

- 3) 布尔函数必须是非退化和 0,1 平衡的布尔函数。

- 4) 布尔函数形式必须清晰、简洁,易于实现。

2.1.4 循环移位常量 R 和 R' 的选取

循环移位常量 R 和 R' 的选取需要满足以下要求:

- 1) 当变量 x 遍历 0~15 时, $R \cdot x \bmod 32$, $R' \cdot x \bmod 32$, $(R+R') \cdot x \bmod 32$ 在 0~31 之间均匀分布,使消息扩散更加均匀。

- 2) 与循环移位常量 S 和 S' 及 P_0 置换相结合,使算法对消息比特的扩散速度加快。

2.1.5 循环移位常量 S 和 S' 的选取

循环移位常量 S 和 S' 的作用是加速消息比特扩散,增加布尔函数 3 个输入变量间的混乱, S 和 S' 的选取需要满足以下要求:

- 1) S 和 S' 差的绝对值在 8 左右,且 S' 为素数, S 为间距较远的奇数,使消息扩散更加均匀。

- 2) 与循环移位常量 R 和 R' 相结合,使算法对消息比特的扩散速度加快。

- 3) 所选的 S 和 S' 便于 8 位智能卡实现。

- 4) S 和 S' 与 P_0 置换的循环移位参数所产生的作用(尤其是雪崩效应)不相互抵消。

2.1.6 加法常量的选取

加法常量起随机化作用。对模 2^{32} 算术加运算而言,加法常量可以减少输入和输出间的线性和差分遗传概率^[9]。对加法常量的选取需要满足以下要求:

- 1) 加法常量的二进制表示中 0,1 基本平衡。

- 2) 加法常量的二进制表示中最长 1 游程小于 5,0 游程小于 4。

- 3) 加法常量的数学表达形式明确,便于记忆。

2.2 消息扩展算法的设计

消息扩展算法将 512 b 的消息分组扩展成 2 176 b 的消息分组。通过线性反馈移位寄存器来实现消息扩展,在较少的运算量下达到较好的扩展效果。消息扩展算法在 SM3 密码杂凑算法中作用主要是加强消息比特之间的相关性,减小通过消息扩展弱点对杂凑算法的攻击可能性。消息扩展算法有以下要求:

- 1) 消息扩展算法满足保熵性;
- 2) 对消息进行线性扩展,使扩展后的消息之间具有良好的相关性;
- 3) 具有较快的雪崩效应;
- 4) 适合软硬件和智能卡实现.

3 SM3 密码杂凑算法的软硬件实现

SM3 密码杂凑算法结构上和 SHA-256 相似,并且链接变量长度、消息分组大小和步数均与 SHA-256 相同. 测试向量和 C 程序代码见附录 A 和附录 B. 本节从 32 b 平台软件性能、ASIC 芯片实现性能和 FPGA 平台实现性能 3 个方面将 SM3 密码杂凑算法和 SHA-256, SHA-512, Whirlpool 与 SHA-3 进行比较.

3.1 软件实现及性能

SM3 密码杂凑算法和 SHA-256, SHA-512, Whirlpool 与 SHA-3 软件实现及性能测试环境如下.

- 1) 处理器: Intel® Core™ i7-4770 @ 3.4 GHz;
- 2) 内存: 8 GB;
- 3) 操作系统: 64 b Windows 7 操作系统;
- 4) 编译环境: Visual Studio 2010.

其中, SHA-256, SHA-512, Whirlpool 与 SHA-3 为 OpenSSL 实现. 测试分别在 Win32 平台和 X64 平台下, 对各种算法在输入长度分别为 16 B, 64 B, 1 024 B 和 8 192 B 的情况下进行, 详细结果如表 1 所示:

表 1 SM3 密码杂凑算法和其他标准软件速度 cycles/B

平台	算法	输入长度			
		16 B	64 B	1 024 B	8 192 B
Win32	SM3	83	40	21	19
	SHA-256	104	44	20	16
	SHA-512	517	131	39	32
	Whirlpool	245	126	71	68
	SHA3-256	450	109	53	50
X64	SM3	63	30	16	15
	SHA-256	76	34	16	14
	SHA-512	149	43	12	9
	Whirlpool	113	60	37	36
	SHA3-256	95	23	10	10

从表 1 可以得出: 在 Win32 和 X64 环境下, 当消息长度为 16 B 时, SM3 密码杂凑算法的软件执行速度高于其他 4 种算法, 速度为 SHA-256 的 125%; 在 Win32 环境下, 当消息长度大于等于 64 B 时, SM3 密码杂凑算法的软件执行速度和 SHA-256 相当, 高于其他 3 种算法; 在 X64 环境下, 当消息长度等于 64 B 时, SM3 密码杂凑算法的软件执行速度和 SHA-256 相当, 低于 SHA-3-256, 高于 SHA-512 和 Whirlpool; 在 X64 环境下, 当消息长度大于 64 B 时, SM3 密码杂凑算法的软件执行速度和 SHA-256 相当, 低于 SHA-512 和 SHA-3-256, 高于 Whirlpool.

3.2 ASIC 芯片实现及性能

SM3 密码杂凑算法和 SHA-256、SHA-512、Whirlpool 与 SHA-3 在 COMS 0.18 μm 工艺下的 ASIC 芯片实现及性能如表 2 所示:

表 2 SM3 密码杂凑算法和其他标准的 ASIC 实现

算法	面积 (gates)	时钟 /MHz	吞吐量 /Mbps	吞吐量面积比 /(Kbps/gate)
SM3 ^[10]	11 068	216.00	1 619	146.28
SHA-256 ^[11]	15 400	189.75	1 349	87.60
SHA-512 ^[11]	30 747	169.20	1 969	64.04
Whirlpool ^[11]	38 911	101.94	2 485	63.86
SHA-3 ^[12]	56 320	487.80	21 229	376.94

从表 2 可以得出: SM3 密码杂凑算法的 ASIC 实现面积要优于其他 4 种算法, 实现面积分别为其他算法的 72%, 36%, 28% 和 20%, 吞吐量介于 SHA-256 和 SHA-512 之间, 吞吐量面积比仅低于 SHA-3, 优于其他 3 种算法. 整体而言, SM3 密码杂凑算法为性能优越的 Merkle-Damgård 结构杂凑算法.

3.3 FPGA 平台上的实现及性能

SM3 密码杂凑算法和 SHA-256 的 FPGA 实现平台为 Xilinx Virtex-5, 其实现及性能如表 3 所示:

表 3 SM3 密码杂凑算法和其他标准的 FPGA 实现

算法	面积 (slices)	时钟 /MHz	吞吐量 /Mbps	吞吐量面积比 /(Mbps/slice)
SM3 ^[13]	234	215	1 619	6.92
SHA-256 ^[14]	319	221	1 714	5.37
SHA-512 ^[14]	605	188	2 188	3.62
Whirlpool ^[11]	7 507 (2 502)	91.37 (126.5)	4 678 (6 477)	0.62 (2.59)
SHA-3 ^[15]	1 272	282.7	12 817	10.07

SM3 密码杂凑算法的 FPGA 实现面积要优于其他 4 种算法, 实现面积分别为其他算法的 73%, 39%, 3% (9%) 和 18%, 吞吐量为 SHA-256 的 94%, 吞吐量面积比仅低于 SHA-3, 略高于 SHA-256. 整体而言, SM3 密码杂凑算法在 Xilinx Virtex-5 上的实现和性能与 SHA-256 相当.

4 SM3 密码杂凑算法的安全性分析

本节给出了 SM3 密码杂凑算法的安全性分析结果. 同时将 SM3 密码杂凑算法和 ISO/IEC 10118-3 标准算法以及部分国家标准杂凑算法根据已有的公开分析结果在安全性上进行比较.

4.1 SM3 密码杂凑算法的安全性分析结果

目前已公开发表的针对 SM3 密码杂凑算法的安全性分析的论文集中在碰撞攻击、原像攻击和区分攻击 3 个方面.

模差分分析方法^[3-5]是寻找杂凑算法碰撞最常用的方法, 一般分析过程可以描述如下: 1) 选择合适的消息差分, 它决定了攻击成功的概率; 2) 针对选择的消息差分寻找可行的差分路线; 3) 推导出保证差分路线可行的充分条件, 在寻找差分路线的过程中, 链接变量的条件被确定下来, 一个可行的差分路线就意味着从路线上推导出来的所有的链接变量的条件相互之间没有冲突; 4) 使用消

息修改技术, 使得被修改的消息满足尽可能多的充分条件. 近年又出现了使用自动化搜索方法寻找差分路线^[16-17]. 针对 SM3 密码杂凑算法的特性, Mendel 等人^[18]在 CT-RSA 2013 上给出了 20 步可实现复杂度的 SM3 密码杂凑算法的碰撞攻击和 24 步可实现复杂度的自由起始碰撞攻击.

Merkle-Damgård 结构杂凑算法的原像攻击主要采用中间相遇攻击^[19-20]及其改进方法, 比如差分中间相遇攻击^[21]等. 寻找原像的过程首先需要寻找单个消息分组的伪原像, 之后使用伪原像转化原像的方法^[22]将伪原像转化为多个分组的原像. 寻找伪原像的过程可以描述如下: 1) 选择合适的独立消息字(或比特), 记为独立消息字 I 和独立消息字 II. 并根据独立消息字将压缩函数分成 3 个部分, 分别记为独立部分 I、独立部分 II 和匹配部分. 其中, 独立消息字 I 和独立部分 II, 独立消息字 II 和独立部分 I 相互独立. 2) 随机设定除独立消息字 I 和 II 之外的其他消息和独立部分 I 和 II 位置的链接变量. 3) 利用独立消息字 I 和独立部分 I 计算列表 L_1 , 利用独立消息字 II 和独立部分 II 计算列表 L_2 . 4) 寻找 L_1 和 L_2 的一个碰撞, 此碰撞对应的初始值和消息即为一个伪原像. 随后又出现了带完全二分结构体^[23]的中间相遇攻击等方法. 带完全二分结构体的中间相遇攻击如图 3 所示, 其中 IW I, IW II 表示独立消息字.

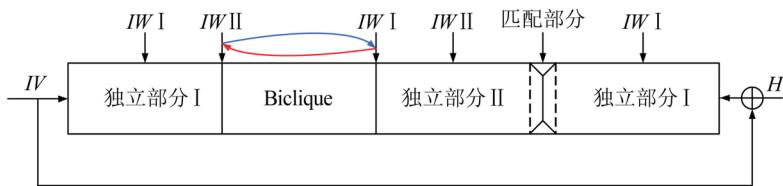


图 3 带完全二分结构体的中间相遇攻击

使用中间相遇攻击方法, Zou 等人^[24]在 ICISC 2011 上给出了从第 1 步开始的 28 步 SM3 密码杂凑算法的原像攻击和从中间开始的 30 步 SM3 密码杂凑算法的原像攻击. 2012 年, Wang 和 Shen^[25]使用差分中间相遇攻击方法给出了 29 步和 30 步 SM3 密码杂凑算法的原像攻击, 同时给出了 31 步和 32 步 SM3 密码杂凑算法的伪原像攻击. 所有的分析结果均从第 1 步开始.

对 SM3 密码杂凑算法的区分攻击主要是使用飞去来器(boomerang)区分攻击, 其主要思想是

使用中间一步或者多步链接变量的衔接将 2 条短的差分路线构造长的差分路线, 进而构造出满足输入输出差分的四元组. 如图 4 所示, 一般过程可以描述如下: 1) 选择合适的消息差分, 构造攻击所需的短差分路线. 消息差分的选取应尽量是充分条件出现在衔接位置附近. 2) 检测衔接位置的充分条件是否矛盾. 3) 随机选择衔接位置的链接变量, 使用消息修改技术, 使得被修改的消息满足尽可能多的充分条件. 4) 从衔接位置开始, 向两端构造相应测差分路线, 进而推导出对应的输入输出差分.

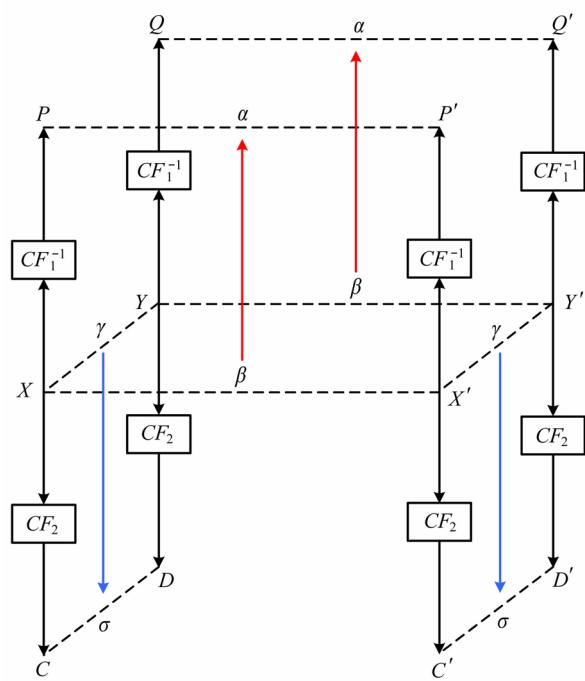


图 4 飞去来器攻击图示

在 SAC 2012 上, Kircanski 等人^[26]给出了 32 步到 35 步 SM3 密码杂凑算法压缩函数的飞去来器区分攻击, 同时给出了 32 步和 33 步的区分攻击实例以及利用 SM3 密码杂凑算法的移位特点, 给出了 SM3-XOR(将 SM3 密码杂凑算法中所有的非线性运算用异或运算代替)的滑动-移位特性。2014 年, Bai 等人^[27]改进了 SM3 密码杂凑算法的飞去来器区分攻击, 给出了 34 步到 37 步飞去来器区分攻击以及 34 步和 35 步的区分攻击实例。SM3 密码杂凑算法的分析结果如表 4 所示:

表 4 SM3 密码杂凑算法的分析结果

攻击类型	目标	步数	复杂度	文献
碰撞攻击	杂凑函数	20	可实现复杂度	[18]
自由起始碰撞攻击	压缩函数	24	可实现复杂度	
原像攻击	杂凑函数	28	$2^{241.5}$	[24]
原像攻击	杂凑函数	30	2^{249}	
原像攻击	杂凑函数	29	2^{245}	
原像攻击	杂凑函数	30	$2^{251.1}$	
伪原像攻击	压缩函数	31	2^{245}	[25]
伪原像攻击	压缩函数	32	$2^{251.1}$	
飞去来器区分攻击	压缩函数	33	可实现复杂度	[26]
飞去来器区分攻击	压缩函数	35	2^{117}	
飞去来器区分攻击	压缩函数	35	可实现复杂度	[27]
飞去来器区分攻击	压缩函数	37	2^{192}	

4.2 SM3 密码杂凑算法和其他杂凑标准对比结果

SM3 密码杂凑算法和其他杂凑标准 SHA-1, SHA-2, RIPEMD-128, RIPEMD-160, Whirlpool, Stribog 和 KECCAK 的安全性分析对比结果如表 5 所示:

表 5 SM3 密码杂凑算法和其他杂凑标准的最好分析结果

算法	攻击类型	步(轮)数	百分比/%	文献
SM3	碰撞攻击	20	31	[18]
	原像攻击	30	47	[24-25]
	区分器攻击	37	58	[27]
SHA-1	碰撞攻击	80	100	[4, 28-29]
	原像攻击	62	77.5	[30]
RIPEMD-128	碰撞攻击	40	62.5	[31]
	原像攻击	36	56.25	[32]
	区分器攻击	64	100	[33]
RIPEMD-160	原像攻击	34	53.12	[34]
	区分器攻击	51	79.68	[35]
SHA-256	碰撞攻击	31	48.4	[36]
	原像攻击	45	70.3	[23]
	区分器攻击	47	73.4	[37]
Whirlpool	碰撞攻击	8	80	[38]
	原像攻击	6	60	[38]
	区分器攻击	10	100	[39]
Stribog	碰撞攻击	7.5	62.5	[40]
	原像攻击	6	50	[41]
KECCAK-256	碰撞攻击	5	20.8	[42]
	原像攻击	2	8	[43]
	区分器攻击	24	100	[44]
KECCAK-512	碰撞攻击	3	12.5	[42]
	区分器攻击	24	100	[44]

从表 5 可以得出: 在碰撞攻击方面, SM3 密码杂凑算法的攻击百分比仅比 KECCAK 高, 比其他杂凑标准低, 但在 MD-SHA 类算法中最低, 仅占总步数的 31%; 在原像攻击方面, SM3 密码杂凑算法的攻击百分比仅比 KECCAK 高, 比其他杂凑标准低, 但在 MD-SHA 类算法中最低, 占总步数的 47%; 在区分器攻击方面, SM3 密码杂凑算法均比其他杂凑标准低, 仅有 58%, 约占总步数的一半左右。这些分析结果体现了 SM3 密码杂凑算法的高安全性。

5 结论

本文介绍了 SM3 密码杂凑算法的设计原理、算法特点、软硬件实现和安全性分析。同国内外其他算法比较结果来看, SM3 密码杂凑算法具有较

高的安全性和软硬件实现面积小,算法实现效率要高于或者等同于国内外其他标准算法。

参 考 文 献

- [1] Kelsey J, Schneier B. Second preimages on n -bit hash functions for much less than 2^n work [G] //LNCS 3494: Proc of the 24th Annual Int Conf on the Theory and Applications of Cryptographic Techniques. Berlin: Springer, 2005: 474-490
- [2] Kelsey J, Kohno T. Herding hash functions and the nostradamus attack [G] //LNCS 4004: Proc of the 24th Annual Int Conf on the Theory and Applications of Cryptographic Techniques. Berlin: Springer, 2006: 183-200
- [3] Wang X, Feng D, Lai X, et al. Collisions for hash functions MD4, MD5, HAVAL-128 and RIPEMD [OL]. 2004 [2016-10-07]. <https://eprint.iacr.org/2004/199.pdf>
- [4] Wang X, Yin Y L, Yu H. Finding collisions in the full SHA-1 [G] //LNCS 3621: Proc of the 25th Annual Int Cryptology Conf. Berlin: Springer, 2005: 17-36
- [5] Wang X, Yu H. How to break MD5 and other hash functions [G] //LNCS 3494: Proc of the 24th Annual Int Conf on the Theory and Applications of Cryptographic Techniques. Berlin: Springer, 2005: 19-35
- [6] National Institute of Standards and Technology. SHA-3 competition [EB/OL]. (2005-04-15) [2016-10-07]. http://csrc.nist.gov/groups/ST/hash/documents/FR_Note_Nov07.pdf
- [7] 国家密码管理局. GM/T 0004—2012 信息安全技术 SM3 密码杂凑算法[S/OL]. 2012 [2016-10-07]. <http://www.oscca.gov.cn/UpFile/20101222141857786.pdf>
- [8] 国家标准化委员会. GB/T32905—2016 信息安全技术 SM3 密码杂凑算法[S/OL] 2016 [2016-10-07]. <http://www.soc.gov.cn/gzfw/ggcx/gjbzgg/201614/>
- [9] Miyano H. Addend dependency of differential/linear probability of addition [J]. IEICE Trans on Fundamentals of Electronics, Communications and Computer Sciences, 1998, E81-A(1): 106-109
- [10] Ao T, He Z, Dai K, et al. A compact hardware implementation of SM3 [C] //Proc of 2014 IEEE Int Conf on Consumer Electronics-China. Piscataway, NJ: IEEE, 2014: 1-4
- [11] Satoh A. ASIC hardware implementations for 512-bit hash function whirlpool [C] //Proc of 2008 IEEE Int Symp on Circuits and Systems. Piscataway, NJ: IEEE, 2008: 2917-2920
- [12] Tillich S, Feldhofer M, Kirschbaum M, et al. High-speed hardware implementations of BLAKE, Blue Midnight Wish, CubeHash, ECHO, Fugue, Grøstl, Hamsi, JH, KECCAK, Luffa, Shabal, SHAvite-3, SIMD, and Skein [OL]. 2009 [2016-10-07]. <http://eprint.iacr.org/2009/510>
- [13] Ma Y, Xia L, Lin J, et al. Hardware performance optimization and evaluation of SM3 hash algorithm on FPGA [G] //LNCS 7618: Proc of the 14th Int Conf on Information and Communications Security. Berlin: Springer, 2012: 105-118
- [14] HELION: Fast SHA-256 core for xilinx FPGA [OL]. 2011 [2016-10-07]. <http://www.helion.com/>
- [15] Homsirikamol E, Rogawski M, Gaj K. Comparing hardware performance of round 3 SHA-3 candidates using multiple hardware architectures in Xilinx and Altera FPGAs [C] //Proc of the ECRYPT II Hash. Tallinn, Estonia, 2011: 19-34
- [16] Cannière C D, Rechberger C. Finding SHA-1 characteristics: General results and applications [G] //LNCS 4284: Proc of the 12th Int Conf on the Theory and Application of Cryptology and Information Security. Berlin: Springer, 2006: 1-20
- [17] Mendel F, Nad T, Schläffer M. Finding SHA-2 characteristics: Searching through a minefield of contradictions [G] //LNCS 7073: Proc of the 17th Int Conf on the Theory and Application of Cryptology and Information Security. Berlin: Springer, 2011: 288-307
- [18] Mendel F, Nad T, Schläffer M. Finding collisions for round-reduced SM3 [G] //LNCS 7779: Proc of the Cryptographers' Track at the RSA Conf 2013. Berlin: Springer, 2013: 174-188
- [19] Aoki K, Sasaki Y. Preimage attacks on one-block MD4, 63-step MD5 and more [G] //LNCS 5381: Proc of the 15th Int Workshop on Selected Areas in Cryptography. Berlin: Springer, 2008: 103-119
- [20] Diffie W, Hellman M E. Special feature exhaustive cryptanalysis of the NBS data encryption standard [J]. Computer, 1977, 10(6): 74-84
- [21] Knellwolf S, Khovratovich D. New preimage attacks against reduced SHA-1 [G] //LNCS 7417: Proc of the 32nd Annual Cryptology Conf. Berlin: Springer, 2012: 367-383
- [22] Menezes A J, van Oorschot P C, Vanstone S A. Handbook of Applied Cryptography [M]. Boca Raton: CRC Press, 1996
- [23] Khovratovich D, Rechberger C, Savelieva A. Bicliques for preimages: Attacks on Skein-512 and the SHA-2 family [G] //LNCS 7549: Proc of the 19th Int Workshop on Fast Software Encryption. Berlin: Springer, 2012: 244-263

- [24] Zou J, Wu W, Wu S, et al. Preimage attacks on step-reduced SM3 hash function [G] //LNCS 7259: Proc of the 14th Int Conf on Information Security and Cryptology. Berlin: Springer, 2011: 375–390
- [25] Wang G, Shen Y. Preimage and pseudo-collision attacks on step-reduced SM3 hash function [J]. Information Processing Letters, 2013, 113(8): 301–306
- [26] Kircanski A, Shen Y, Wang G, et al. Boomerang and slide-rotational analysis of the SM3 hash function [G] //LNCS 7707: Proc of the 19th Int Conf on Selected Areas in Cryptography. Berlin: Springer, 2012: 304–320
- [27] Bai D, Yu H, Wang G, et al. Improved boomerang attacks on round-reduced SM3 and keyed permutation of BLAKE-256 [J]. IET Information Security, 2015, 9(3): 167–178
- [28] Wang X, Yao A C, Yao F. Cryptanalysis on SHA-1 [OL]. 2005 [2016-10-07]. http://csrc.nist.gov/groups/ST/hash/documents/Wang_SHA1-New-Result.pdf
- [29] Stevens M. New collision attacks on SHA-1 based on optimal joint local-collision analysis [G] //LNCS 7881: Proc of the 32nd Annual Int Conf on the Theory and Applications of Cryptographic Techniques. Berlin: Springer, 2013: 245–261
- [30] Espitau T, Fouque P-A, Karpman P. Higher-order differential meet-in-the-middle preimage attacks on SHA-1 and BLAKE [G] //LNCS 9215: Proc of the 35th Annual Cryptology Conf. Berlin: Springer, 2015: 683–701
- [31] Wang G. Practical collision attack on 40-step RIPEMD-128 [G] //LNCS 8366: Proc of the Cryptographer's Track at the RSA Conf 2014. Berlin: Springer, 2014: 444–460
- [32] Wang L, Sasaki Y, Komatsubara W, et al. Preimage attacks on step-reduced RIPEMD/RIPEMD-128 with a new local-collision approach [G] //LNCS 6558: Proc of the Cryptographers' Track at the RSA Conf 2011. Berlin: Springer, 2011: 197–212
- [33] Landelle F, Peyrin T. Cryptanalysis of full RIPEMD-128 [G] //LNCS 7881: Proc of the 32nd Annual Int Conf on the Theory and Applications of Cryptographic Techniques. Berlin: Springer, 2013: 228–244
- [34] Wang G, Shen Y. (Pseudo-) Preimage attacks on step-reduced HAS-160 and RIPEMD-160 [G] //LNCS 8783: Proc of the 17th Int Conf on Information Security. Berlin: Springer, 2014: 90–103
- [35] Sasaki Y, Wang L. Distinguishers beyond three rounds of the RIPEMD-128/-160 compression functions [G] //LNCS 7341: Proc of the 10th Int Conf on Applied Cryptography and Network Security. Berlin: Springer, 2012: 275–292
- [36] Mendel F, Nad T, Schlaffer M. Improving local collisions: New attacks on reduced SHA-256 [G] //LNCS 7881: Proc of the 32nd Annual Int Conf on the Theory and Applications of Cryptographic Techniques. Berlin: Springer, 2013: 262–278
- [37] Biryukov A, Lamberger M, Mendel F, et al. Second-order differential collisions for reduced SHA-256 [G] //LNCS 7073: Proc of the 17th Annual Int Conf on the Theory and Application of Cryptology and Information Security. Berlin: Springer, 2011: 270–287
- [38] Sasaki Y, Wang L, Wu S, et al. Investigating fundamental security requirements on Whirlpool: improved preimage and collision attacks [G] //LNCS 7658: Proc of the 18th Int Conf on the Theory and Application of Cryptology and Information Security. Berlin: Springer, 2012: 562–579
- [39] Lamberger M, Mendel F, Rechberger C, et al. The rebound attack and subspace distinguishers: Application to Whirlpool [OL]. 2010 [2016-10-07]. <http://eprint.iacr.org/2010/198.pdf>
- [40] Ma B, Li B, Hao R, et al. Improved cryptanalysis on reduced-round GOST and Whirlpool hash function [G] //LNCS 8479: Proc of the 12th Int Conf on Applied Cryptography and Network Security. Berlin: Springer, 2014: 289–307
- [41] AlTawy R, Youssef A M. Preimage attacks on reduced-round Stribog [G] //LNCS 8469: Proc of the 7th Int Conf on Cryptology in Africa. Berlin: Springer, 2014: 109–12
- [42] Dinur I, Dunkelman O, Shamir A. Collision attacks on up to 5 rounds of SHA-3 using generalized internal differentials [G] //LNCS 8424: Proc of the 20th International Workshop on Fast Software Encryption. Berlin: Springer, 2013: 219–240
- [43] Homsirikamol E, Morawiecki P, Rogawski M, et al. Security margin evaluation of SHA-3 contest finalists through SAT-based attacks [G] //LNCS 7564: Proc of the 11th IFIP TC 8 Int Conf on Computer Information Systems and Industrial Management. Berlin: Springer, 2012: 56–67
- [44] Duan M, Lai X. Improved zero-sum distinguisher for full round Keccak-f permutation [J]. Chinese Science Bulletin, 2012, 57(6): 694–697



王小云

教授,博士生导师,主要研究方向为密码理论与技术.

xiaoyunwang@mail.tsinghua.edu.cn



于红波

博士,副教授,主要研究方向为密码算法分析与设计.

yuhongbo@mail.tsinghua.edu.cn

附录 A. SM3 密码杂凑算法测试向量.

SM3 密码杂凑算法的测试向量如下：

A.1 测试向量 1

输入消息为“abc”，其 ASCII 码表示为“616263”。

填充后的消息：

```
61626380 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000018
```

杂凑值：

```
66c7f0f4 62eedd9 d1f2d46b dc10e4e2
4167c487 5cf2f7a2 297da02b 8f4ba8e0
```

A.2 测试向量 2

输入 512 b 消息：

```
61626364 61626364 61626364 61626364
61626364 61626364 61626364 61626364
61626364 61626364 61626364 61626364
61626364 61626364 61626364 61626364
```

填充后的消息：

```
61626364 61626364 61626364 61626364
61626364 61626364 61626364 61626364
61626364 61626364 61626364 61626364
61626364 61626364 61626364 61626364
```

```
80000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
80000000 00000000 00000000 00000000
00000000 00000000 00000000 00000200
```

杂凑值：

```
debe9ff9 2275b8a1 38604889 c18e5a4d
6fdb70e5 387e5765 293dcba3 9c0c5732
```

附录 B. SM3 密码杂凑算法 C 程序代码.

SM3 密码杂凑算法的 C 程序代码清单如下：

```
///////////////
```

/// 类型重定义

```
///////////////
```

```
typedef unsigned int u32;
```

```
typedef unsigned char u8;
```

```
typedef unsigned __int64 u64;
```

```
///////////////
```

/// 以下 7 个函数为轮函数中用到的函数

```
///////////////
```

```
#define FF1(X,Y,Z) (X^Y^Z)
```

```
#define FF2(X,Y,Z) ((X&Y) | (X&Z) |
(Y&Z))
#define GG1(X,Y,Z) (X^Y^Z)
#define GG2(X,Y,Z) ((X&Y) | ((~X)
&Z))
#define ROTL32(X,num) ((X<<num) |
(X>>(32-num)))
#define P0(X) ((X)^ROTL32((X),9)^
ROTL32((X),17))
#define P1(X) ((X)^ROTL32((X),15)^
ROTL32((X),23))
//将 32 b 的数所包含的 4 个字节顺序翻转
#define REVERSE32(w,x) {\
    u32 tmp=(w);\
    tmp=(tmp>>16) | (tmp<<16);\
    (x)=(((tmp&0xff00ff00)>>8) |\
    ((tmp&0x00ff00ff)<<8));\
}
typedef struct {
    u32 state[8];
    u64 bitcount;
    u32 buffer[24];
}SM3_256_CTX;
//////////////Set Init Value/////////////
const u32 T[64]={0x79cc4519, 0xf3988a32,
0xe7311465, 0xce6228cb, 0x9cc45197, 0x3988a32f,
0x7311465e, 0xe6228cbc, 0xcc451979, 0x988a32f3,
0x311465e7, 0x6228cbce, 0xc451979c, 0x88a32f39,
0x11465e73, 0x228cbce6, 0x9d8a7a87, 0x3b14f50f,
0x7629ea1e, 0xec53d43c, 0xd8a7a879, 0xb14f50f3,
0x629ea1e7, 0xc53d43ce, 0x8a7a879d, 0x14f50f3b,
0x29ea1e76, 0x53d43cec, 0xa7a879d8, 0x4f50f3b1,
0x9ea1e762, 0x3d43cec5, 0x7a879d8a, 0xf50f3b14,
0xea1e7629, 0xd43cec53, 0xa879d8a7, 0x50f3b14f,
0xae1e7629e, 0x43cec53d, 0x879d8a7a, 0xf3b14f5,
0x1e1e7629ea, 0x3cec53d4, 0x79d8a7a8, 0xf3b14f50,
0xe7629ea1, 0xec53d43, 0x9d8a7a87, 0x3b14f50f,
0x7629ea1e, 0xec53d43c, 0xd8a7a879, 0xb14f50f3,
0x629ea1e7, 0xc53d43ce, 0x8a7a879d, 0x14f50f3b,
0x29ea1e76, 0x53d43cec, 0xa7a879d8, 0x4f50f3b1,
0x9ea1e762, 0x3d43cec5};
```

//将中间状态值初始化为 IV

```

void SM3_256_Init(SM3_256_CTX * ctx){
    u32 IH[8]={0x7380166f, 0x4914b2b9,
               0x172442d7, 0xda8a0600, 0xa96f30bc,
               0x163138aa, 0xe38dee4d, 0xb0fb0e4e};
    int i;
    for (i=0; i < 8; i++){
        ctx->state[i]=IH[i];
    }
    memset(ctx->buffer,0,sizeof(u32) * 16);
    ctx->bitcount=0;
}

///////////
/// SM3_256_Block 为处理一个 512 b 分块
/// 的过程,最后更新 ctx 中的 state 状态值
/////////
void SM3_256_Block(SM3_256_CTX * ctx){
    u32 A,B, C, D, E, F, G, H,temp;
    u32 SS1, SS2, TT1,TT2,Const;
    u32 W[68]={0};
    int i;
    u32 t;
    u32 t1, t2=0x7a879d8a;
    A=ctx->state[0];
    B=ctx->state[1];
    C=ctx->state[2];
    D=ctx->state[3];
    E=ctx->state[4];
    F=ctx->state[5];
    G=ctx->state[6];
    H=ctx->state[7];
}

//////Expand Message Block(消息扩展)/////
W[0]=ctx->buffer[0];
W[1]=ctx->buffer[1];
W[2]=ctx->buffer[2];
W[3]=ctx->buffer[3];
W[4]=ctx->buffer[4];
W[5]=ctx->buffer[5];
W[6]=ctx->buffer[6];
W[7]=ctx->buffer[7];
W[8]=ctx->buffer[8];
W[9]=ctx->buffer[9];
W[10]=ctx->buffer[10];
W[11]=ctx->buffer[11];

W[12]=ctx->buffer[12];
W[13]=ctx->buffer[13];
W[14]=ctx->buffer[14];
W[15]=ctx->buffer[15];
for (i=16; i<68; i++){
    temp=W[i-16]^W[i-9]^
        ROTL32(W[i-3],15);
    W[i]=P1(temp)^ROTL32(W[i-13],
                           7)^W[i-6];
}
//////Compression Function(压缩函数)/////
for (i=0; i<64; i++){
    if (i<16){
        Const=0x79cc4519;
        SS1=ROTL32(ROTL32(A,12)+E+ROTL32(Const, i), 7);
        SS2=SS1^ROTL32(A,12);
        TT1=FF1(A,B,C)+D+SS2+
            (W[i]^W[i+4]);
        TT2=GG1(E,F,G)+H+SS1+
            W[i];
    }
    else{
        Const=0x7a879d8a;
        SS1=ROTL32(ROTL32(A,12)+E+ROTL32(Const, i), 7);
        SS2=SS1^ROTL32(A,12);
        TT1=FF2(A,B,C)+D+SS2+
            (W[i]^W[i+4]);
        TT2=GG2(E,F,G)+H+SS1+
            W[i];
    }
    D=C;
    C=ROTL32(B,9);
    B=A;
    A=TT1;
    H=G;
    G=ROTL32(F,19);
    F=E;
    E=P0(TT2);
}
//更新状态值
ctx->state[0]^=A;

```

```
ctx->state[1]=B;
ctx->state[2]=C;
ctx->state[3]=D;
ctx->state[4]=E;
ctx->state[5]=F;
ctx->state[6]=G;
ctx->state[7]=H;
}

//////////SM3 256 Hash function///////////
/// 定义 SM3 完整 hash 过程
/// Inmessage 代表输入的消息, MessageLen
为消息长度,单位为 B
/// OutDigest 代表最后的摘要, DigestLen
为摘要长度,单位为 B
///////////
void SM3( unsigned char * InMessage, int
MessageLen, unsigned char * OutDigest, int *
DigestLen){
    u32 * p_data=(u32 *)InMessage;
    int i,j;
    SM3_256_CTX ctx;
    u32 * p_out=(u32 *) OutDigest;
    SM3_256_Init(&ctx);
    if (MessageLen==0)
        return;
    while(MessageLen>=64){
        //处理前面不用补位的 512 b 块
        for (i=0; i < 16; i++){
            REVERSE32(* p_data,ctx.buffer[i]);
            p_data++;
        }
        SM3_256_Block(&ctx);
        ctx.bitcount+=512;
        MessageLen-=64;
    }
    ///////////padding(填充)///////////
    for (i=0; i < (MessageLen/4); i++){
        REVERSE32(* p_data,ctx.buffer[i]);
        p_data++;
    }
    MessageLen-=4*i;
    ctx.bitcount+=32*i;
    switch(MessageLen){

        case 0:
            ctx.buffer[i]=(0x80)<<24;
            break;
        case 1:
            ctx.buffer[i]=((*(u8 *)p_data)
            <<24)|(0x80<<16);
            ctx.bitcount+=8;
            break;
        case 2:
            ctx.buffer[i]=((*(u8 *)p_data)
            <<24)|((*(u8 *)p_data+1))
            <<16)|(0x80<<8);
            ctx.bitcount+=16;
            break;
        case 3:
            ctx.buffer[i]=((*(u8 *)p_data)
            <<24)|((*(u8 *)p_data+1))
            <<16)|((*(u8 *)p_data+2))
            <<8)|0x80;
            ctx.bitcount+=24;
            break;
    }
    memset(&ctx.buffer[i+1],0,
    sizeof(u32)*(15-i));
    if (i < 56){
        ctx.buffer[14]=ctx.bitcount>>32;
        ctx.buffer[15]=
            ctx.bitcount&0xFFFFFFFF;
        SM3_256_Block(&ctx);
    }
    else{
        SM3_256_Block(&ctx);
        memset(ctx.buffer,0,sizeof(u32)*16);
        *(u64 *)(&ctx.buffer[15])=
            ctx.bitcount;
    }
    //memcpy(OutDigest,ctx.state,sizeof(u32)
    * 8);
    for (i=0; i<8; i++){
        REVERSE32(ctx.state[i],* p_out);
        p_out++;
    }
    * DigestLen=32;
}
```