第 11 章 SHA-1 算法

11.1 算法原理

在安全应用中使用的 Hash 函数称为密码学 Hash 函数。常用的几种密码学 Hash 函数有 MD 系列、SHA 系列等。安全 Hash 算法 SHA（Security Hash Algorithm）是使用最为广泛的 Hash 函数，由美国标准与技术研究所（NIST）设计，并于 1993 年作为联邦信息处理标准（FIPS 180）发布，该版本称为 SHA-0，修订版于 1995 年发布（FIPS 180），通常称之为 SHA-1，即安全 Hash 标准。

11.1.1 SHA-1 总体结构

SHA-1 算法要求输入消息长度小于2^{64}位，将输入消息按 512 位分组进行处理，输出长度为 160 位。图 11-1 显示了处理消息、输出摘要的总体过程。算法过程如下：

第一步：位填充。填充消息使其比特长度满足$n \equiv 448 \pmod{512}$，填充由一个 1 和若干个 0 组成；

第二步：长度填充。用 64 位表示消息位填充前的长度，将其附加在位填充的消息后面；

第三步：初始化缓冲区；

第四步：以 512 位分组为单位处理消息；

第五步：输出结果。

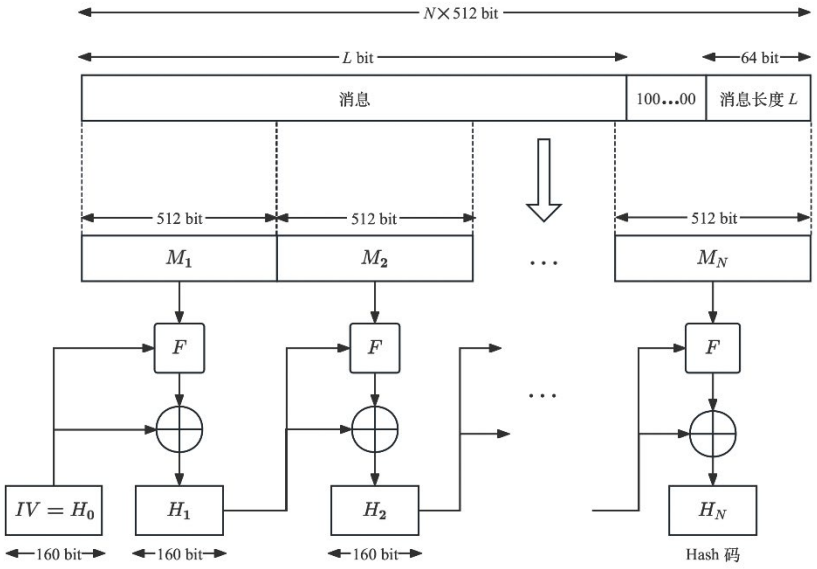

图 11-1 SHA-1 生成消息摘要

11.1.2 SHA-1 详细结构

1. SHA-1 的分组

对于任意长度的消息，首先需要对消息添加位数，使消息总长度模 512 与 448 同余。在消息后添加位数的方法是第一位是 1，其余都是 0。之后将原始的长度（没有添加位数以前的消息比特长度）以 64 位表示，附加于消息后，此时的消息长度正好是 512 位的倍数。SHA-

1 的原始消息长度不能超过2^{64}。另外，SHA-1 的消息长度从低位开始填充。对填充后的消息按 512 位的长度进行分组，表示为$Y_0, Y_1, \ldots, Y_{L-1}$。

对于 512 位的消息分组，SHA-1 将其再分成 16 个子消息分组，每个子消息分组为 32 位，使用$M[k](k = 0, 1, \ldots, 15)$表示。之后将 16 个子消息分组扩充到 80 个子消息分组进行后续计算，记为$W[k](k = 0, 1, \ldots, 79)$，扩充方法如下。

$W_t = M_t$，当$0 \le t \le 15$。

$W_t = (W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) \lll_{32} 1$，当$16 \le t \le 79$。

2. SHA-1 的 4 轮运算

SHA-1 有 4 轮运算，每一轮包括 20 个步骤（共 80 步），最后产生 160 位摘要，如图 11-2 所示。

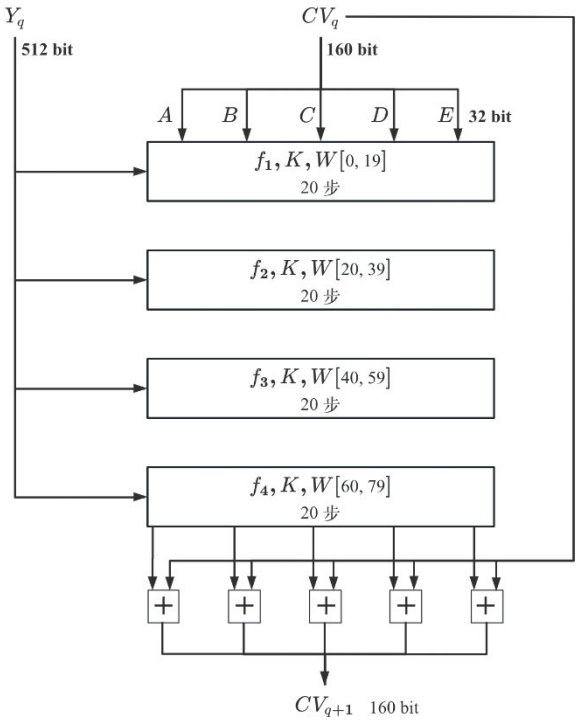

图 11-2 SHA-1 对单个 512 分组的处理

160 位摘要存放在 5 个 32 位的链接变量中，分别标记为A、B、C、D、E，初始值以 16 进制表示如下：

$$A = 0x67452301$$
$$B = 0xEFCDAB89$$
$$C = 0x98BADCFE$$
$$D = 0x10325476$$
$$E = 0xC3D2E1F0$$

当第 1 轮运算中的第 1 步开始处理时，A, B, C, D, E五个链接变量中的值先赋值到另外 5 个记录单元A', B', C', D', E'中。这 5 个值将保留，用于在第 4 轮的最后一个步骤完成之后与链接变量A, B, C, D, E进行求和操作。SHA-1 的 4 轮运算，共 80 步使用同一个函数，该函数的内部结构如图 11-3 所示，表示为如下形式：

$$A, B, C, D, E \leftarrow [(A \lll_{32} 5) + f_t(B, C, D) + E + W_t + K_t], A, (B \lll_{32} 30), C, D$$

其中 $f_t(B,C,D)$为逻辑函数，W_t为子明文分组$W[t]$，K_t为固定常数。

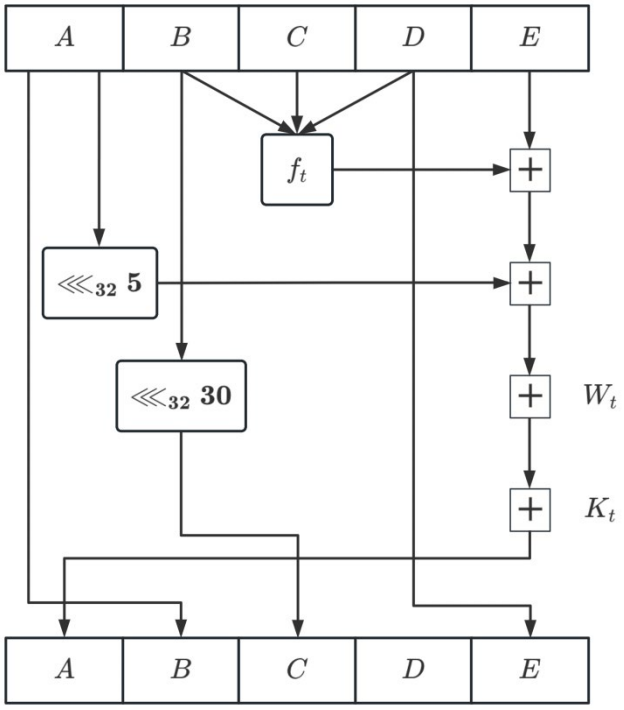

图 11-3 SHA-1 的单轮操作

具体这个函数的意义为：

（1）将$[(A \lll_{32} 5) + f_t(B,C,D) + E + W_t + K_t]$的结果赋值给链接变量$A$；

（2）将链接变量A初始值赋值给链接变量B；

（3）将链接变量B初始值循环左移 30 位赋值给链接变量C；

（4）将链接变量C初始值赋值给链接变量D；

（5）将链接变量D初始值赋值给链接变量E。

3. SHA-1 的逻辑函数

SHA-1 的逻辑函数如表 11-1 所示。

表 11-1 SHA-1 的逻辑函数

轮	步骤	函数定义
1	$0 \le t \le 19$	$f_t(B,C,D) = (B \wedge C) \vee (\overline{B} \wedge D)$
2	$20 \le t \le 39$	$f_t(B,C,D) = B \oplus C \oplus D$
3	$40 \le t \le 59$	$f_t(B,C,D) = (B \wedge C) \vee (B \wedge D) \vee (C \wedge D)$
4	$60 \le t \le 79$	$f_t(B,C,D) = B \oplus C \oplus D$

在操作程序中需要使用固定常数$K_i(i = 0,1,2,\ldots,79)$，取值如表 11-2 所示。

表 11-2 固定常数K_t的取值

轮	步骤	函数定义
1	$0 \le t \le 19$	$K_t = 0x5A827999$
2	$20 \le t \le 39$	$K_t = 0x6ED9EBA1$
3	$40 \le t \le 59$	$K_t = 0x8F1BBCDC$
4	$60 \le t \le 79$	$K_t = 0xCA62C1D6$

11.2 算法伪代码

11.2.1 摘要生成函数

摘要生成函数$sha1_digest$输入明文消息的编码msg，输出产生的摘要$digest$。伪代码如下所示：

算法 11.2.1 $\boldsymbol{sha1\_digest(msg)}$

// 输入：消息msg的编码
// 输出：消息msg的哈希值$digest$
$digest \leftarrow \varepsilon$
$K \leftarrow [0x5A827999, 0x6ED9EBA1, 0x8F1BBCDC, 0xCA62C1D6]$
$H \leftarrow [0x67452301, 0xEFCDAB89, 0x98BADCFE, 0x10325476, 0xC3D2E1F0]$
$EM \leftarrow sha1\_padding(msg)$
将EM按照 512 bit 一组分为l组
for $i \leftarrow 0$ **to** $l-1$ **do**
    $M[i] \leftarrow EM$中第 i 个分组
    $W \leftarrow sha1\_extend(M[i])$
    $sha1\_round(W, H, K)$
$digest \leftarrow H[0] \,||\, H[1] \,||\, H[2] \,||\, H[3] \,||\, H[4]$
return $digest$

11.2.2 消息填充函数

消息填充函数$sha1_padding$负责把消息填充到需要的长度，并且写入消息的长度信息，附加完的消息二进制长度应该为 512 的整数倍。消息填充函数$sha1_padding$输入原始消息的编码msg，输出 16 进制经填充的消息EM。伪代码如下所示：

算法 11.2.2 $\boldsymbol{sha1\_padding(msg)}$

// 输入：需要填充的消息msg
// 输出：填充后的EM
将消息转换成二进制串bin_text
$init_len \leftarrow bin_text$的比特长度
// 填充
$bin\_text \leftarrow bin\_text \,||\, 0b1$
$bin\_len \leftarrow init\_len + 1$
while $bin\_len \bmod 512 \neq 448$ **do**
    $bin\_text \leftarrow bin\_text \,||\, 0b0$
    $bin\_len \leftarrow bit\_len + 1$
将$init_len$用 64 位的格式补充在bin_text最后
$EM \leftarrow$ 将bin_text转为十六进制串
return EM

11.2.3 字扩展函数

字扩展函数$sha1_extend$输入M，输出经扩展的字节分组W。伪代码如下所示：

算法 11.2.3 $\boldsymbol{sha1\_extend(M)}$

// 输入：需要扩充的一组消息M
// 输出：扩展后的 80 个子分组W
$W[0 \ldots 15] \leftarrow M$
for $i \leftarrow 16$ **to** 79 **do**
    $W[i] \leftarrow W[i-3] \oplus W[i-8] \oplus W[i-14] \oplus W[i-16]$
    $W[i] \leftarrow W[i] \lll_{32} 1$
return W

11.2.4 轮函数

轮函数$sha1_round$输入经扩展的字节分组W、数组H和参数K，输出迭代更新后的数组H。伪代码如下所示：

算法 11.2.4 $\boldsymbol{sha1\_round(W, H, K)}$

// 输入：子分组 W、数组H、参数K
// 输出：迭代更新后的数组H
$a, b, c, d, e \leftarrow H[0], H[1], H[2], H[3], H[4]$
for $j \leftarrow 0$ **to** 79 **do**
    $temp \leftarrow a \lll_{32} 5 + sha1\_f(b, c, d, j) + e + W[j] + K[\lfloor j/20 \rfloor]$
    $e \leftarrow d$
    $d \leftarrow c$
    $c \leftarrow b \lll_{32} 30$
    $b \leftarrow a$
    $a \leftarrow temp \mod 2^{32}$
$H[0] \leftarrow (H[0] + a) \mod 2^{32}$
$H[1] \leftarrow (H[1] + b) \mod 2^{32}$
$H[2] \leftarrow (H[2] + c) \mod 2^{32}$
$H[3] \leftarrow (H[3] + d) \mod 2^{32}$
$H[4] \leftarrow (H[4] + e) \mod 2^{32}$
return H

11.2.5 逻辑函数

逻辑函数$sha1_f$在 80 轮中各不相同，$sha1_f$函数具体如下：

算法 11.2.5 $\boldsymbol{sha1\_f(x, y, z, r)}$

// 输入：逻辑函数输入数据x, y, z，当前轮次r
// 输出：逻辑函数输出res
if $r < 20$ **then**
    $res \leftarrow (x \wedge y) \vee (\overline{x} \wedge z)$
else if $r < 40$ **then**
    $res \leftarrow x \oplus y \oplus z$
else if $r < 60$ **then**
    $res \leftarrow (x \wedge y) \vee (x \wedge z) \vee (y \wedge z)$
else
    $res \leftarrow x \oplus y \oplus z$
return res

第 15 章 SM3 算法

SM3 密码杂凑算法由中国国家密码管理局在 2010 年发布，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)。SM3 适用于密码应用中的数字签名和验证、消息认证码的生成与验证以及随机数的生成，可满足多种密码应用的安全需求。

15.1 算法原理

15.1.1 SM3 总体结构

SM3 算法和 MD5 的迭代过程类似，采用 Merkle-Damgard 结构，是在 SHA-256 基础上改进实现的一种算法，其安全性和 SHA-256 相当。SM3 算法的消息分组长度为 512 位，摘要值长度为 256 位，对长度为$l(l < 2^{64})$比特的消息m，SM3 杂凑算法经过填充和迭代压缩，生成杂凑值，杂凑值长度为 256 比特。SM3 算法总体流程如图 15-1 所示。

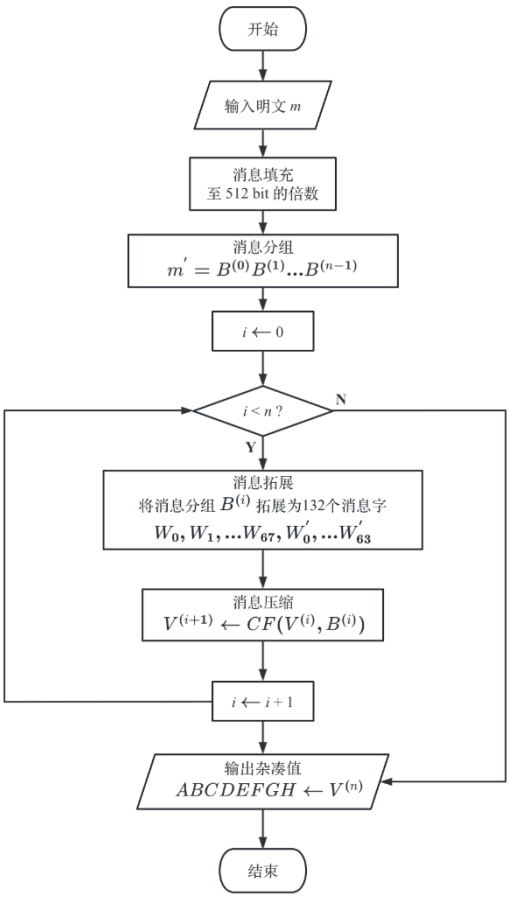

图 15-1 SM3 总体流程图

15.1.2 SM3 详细结构

1. 常量与函数

（1）初始值

初始值IV由 8 个 32 比特字构成，其 16 进制表示如下所示。

$$IV = 0x7380166f\ 4914b2b9\ 172442d7\ da8a0600\ a96f30bc\ 163138aa\ e38dee4d\ b0fb0e4e$$

（2）常量

$$T_j = \begin{cases} 0x79cc4519 & 0 \le j \le 15 \\ 0x7a879d8a & 16 \le j \le 63 \end{cases}$$

（3）布尔函数

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

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

式中X、Y、Z是 32 比特字，$\overline{X}$表示取反运算。

（4）置换函数

$$P_0(X) = X \oplus (X \lll_{32} 9) \oplus (X \lll_{32} 17)$$

$$P_1(X) = X \oplus (X \lll_{32} 15) \oplus (X \lll_{32} 23)$$

式中X是 32 比特字。

2. 消息填充

假设消息m的长度为l比特，则首先将比特“1”添加到消息的末尾，再添加k个“0”，k是满足$l + 1 + k \equiv 448 \pmod{512}$的最小的非负整数。之后再添加一 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$l = 24$，经填充得到比特串：

$$01100001\ 01100010\ 01100011\ 1\ \overbrace{00\ldots00}^{423\text{ 比特}}\ \overbrace{\underbrace{00\ldots011000}_{l\text{的二进制表示}}}^{64\text{ 比特}}$$

3. 迭代压缩

（1）迭代过程

将填充后的消息m'按 512 比特进行分组：

$$m' = B^{(0)}B^{(1)} \ldots B^{(n-1)}$$

其中$n = (l + k + 65)/512$。

对m'按下列方式迭代：

for $i = 0$ **to** $(n-1)$ **do**

$V^{(i+1)} = CF\left(V^{(i)}, B^{(i)}\right)$

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

（2）消息扩展

将消息分组$B^{(i)}$按以下方法扩展生成 132 个消息字$W_0, W_1, \ldots, W_{67}, W_0', \ldots, W_{63}'$，用于压缩函数$CF$：

第一步，将消息分组$B^{(i)}$划分为 16 个字$W_0, W_1, \ldots, W_{15}$。

第二步，

for $j = 16$ **to** 67 **do**

$$W_j \leftarrow P_1\left(W_{j-16} \oplus W_{j-9} \oplus \left(W_{j-3} \lll_{32} 15\right)\right) \oplus \left(W_{j-13} \lll_{32} 7\right) \oplus W_{j-6}$$

第三步，

for $j = 0$ **to** 63 **do**

$$W_j' = W_j \oplus W_{j+4}$$

（3）压缩函数

令A，B，C，D，E，F，G，H为字寄存器，$SS1$，$SS2$，$TT1$，$TT2$为中间变量，压缩函数$V^{(i+1)} = CF\left(V^{(i)}, B^{(i)}\right), 0 \le i \le n-1$。计算过程描述如下：

$ABCDEFGH \leftarrow V^{(i)}$

for $j = 0$ **to** 63 **do**

$$SS1 \leftarrow \left((A \lll_{32} 12) + E + \left(T_j \lll_{32} (j \bmod 32)\right)\right) \lll_{32} 7$$

$$SS2 \leftarrow SS1 \oplus (A \lll_{32} 12)$$

$$TT1 \leftarrow FF_j(A, B, C) + D + SS2 + W_j'$$

$$TT2 \leftarrow GG_j(E, F, G) + H + SS1 + W_j$$

$$D \leftarrow C$$

$$C \leftarrow B \lll_{32} 9$$

$$B \leftarrow A$$

$$A \leftarrow TT1$$

$$H \leftarrow G$$

$$G \leftarrow F \lll_{32} 19$$

$$F \leftarrow E$$

$$E \leftarrow P_0(TT2)$$

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

其中，字的存储为大端（big-endian），左边为高有效位，右边为低有效位。

4. 输出杂凑值

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

输出 256 比特的杂凑值$y = ABCDEFGH$。

15.2 算法伪代码

15.2.1 摘要生成算法

摘要生成算法$sm3_digest$输入消息msg，输出摘要值$digest$。其中状态数据$state$的初始值IV为 8 个 32 比特字，IV的取值见 15.1.2 章节常量与函数中。

算法 15.2.1 $\boldsymbol{sm3\_digest(msg)}$

// 输入：消息msg
// 输出：摘要值$digest$
$state \leftarrow$初始值IV
$msg \leftarrow sm3\_padding(msg)$
$msg_len \leftarrow msg$的字节长度
for $i = 0$ **to** $msg\_len/64$ **do**
    $W, W' \leftarrow sm3_extend(msg$的第$i$个分组$)$
    $state \leftarrow sm3\_round(W, W', state)$
$digest \leftarrow state[0] \,||\, state[1] \,||\, \cdots \,||\, state[7]$
return $digest$

15.2.2 消息填充算法

消息填充算法$sm3_padding$负责把消息填充到需要的长度，填充后的消息二进制长度应该为 512 的整数倍。算法输入消息msg，输出填充后的消息M。算法伪代码如下：

算法 15.2.2 $\boldsymbol{sm3\_padding(msg)}$

// 输入：消息msg
// 输出：填充后的消息M
$msg_len \leftarrow msg$字节长度(使用 64 比特长存储)
$pad\_num \leftarrow 64 - ((msg\_len + 8) \bmod 64)$
$padding \leftarrow (pad_num - 1)$长度的全 0 字节串
$M \leftarrow msg \,||\, 0x80 \,||\, padding \,||\, (8 \cdot msg\_len)$
return M

15.2.3 消息扩展算法

消息扩展算法$sm3_extend$函数将当前组的消息msg进行扩展，输出当前组的消息的扩展结果W和W'，其中的置换函数P_1见 15.1.2 章节常量与函数中。算法伪代码如下：

算法 15.2.3 $\boldsymbol{sm3\_extend(msg)}$

// 输入：64 字节长度的消息分组msg
// 输出：字扩展结果W和W'
$W[0 \cdots 15] \leftarrow msg$
$W' \leftarrow \varepsilon$
for $i = 16$ **to** 67 **do**
    $t \leftarrow P_1(W[i-16] \oplus W[i-9] \oplus (W[i-3] \lll_{32} 15))$
    $t \leftarrow t \oplus (W[i-3] \lll_{32} 15) \oplus W[i-6]$
    $W[i] \leftarrow t$
for $i = 0$ **to** 63 **do**
    $W'[i] \leftarrow W[i] \oplus W[i+4]$
return W, W'

15.2.4 压缩轮函数

迭代压缩函数$sm3_round$输入消息扩展结果W和W'及上一轮的状态数据$state$，输出本轮压缩后的状态数据，其中的常量T_j、布尔函数FF_j和GG_j、置换函数P_0见 15.1.2 章节中的常量与函数部分。算法伪代码如下：

算法 15.2.4 $\boldsymbol{sm3\_round(W, W', state)}$

// 输入：消息扩展结果W和W'，状态数据$state$
// 输出：更新后的状态输出
$A, B, C, D, E, F, G, H \leftarrow state$
for $j = 0$ **to** 63 **do**
    $SS1 \leftarrow ((A \lll_{32} 12) + E + (T_j \lll_{32} (j \bmod 32))) \lll_{32} 7$
    $SS2 \leftarrow SS1 \oplus (A \lll_{32} 12)$
    $TT1 \leftarrow FF_j(A, B, C) + D + SS2 + W'[i]$
    $TT2 \leftarrow GG_j(E, F, G) + H + SS1 + W[i]$
    $D \leftarrow C$
    $C \leftarrow B \lll_{32} 9$
    $B \leftarrow A$
    $A \leftarrow TT1$
    $H \leftarrow G$
    $G \leftarrow F \lll_{32} 19$
    $F \leftarrow E$
    $E \leftarrow P_0(TT2)$
return $ABCDEFGH \oplus state$