Algorithms

Bit-wise Operations

SHA256

Research

InfoSec Write Ups
SHA-2 Wikipedia

Algorithm

Creating Block

M = Message
L = Length$_{M}$
P = Padding
n = Length$_{block}$ / 512

L + P + 64 = 512n
P = 512(ceil((L + 64) / 512))) - (L + 64)

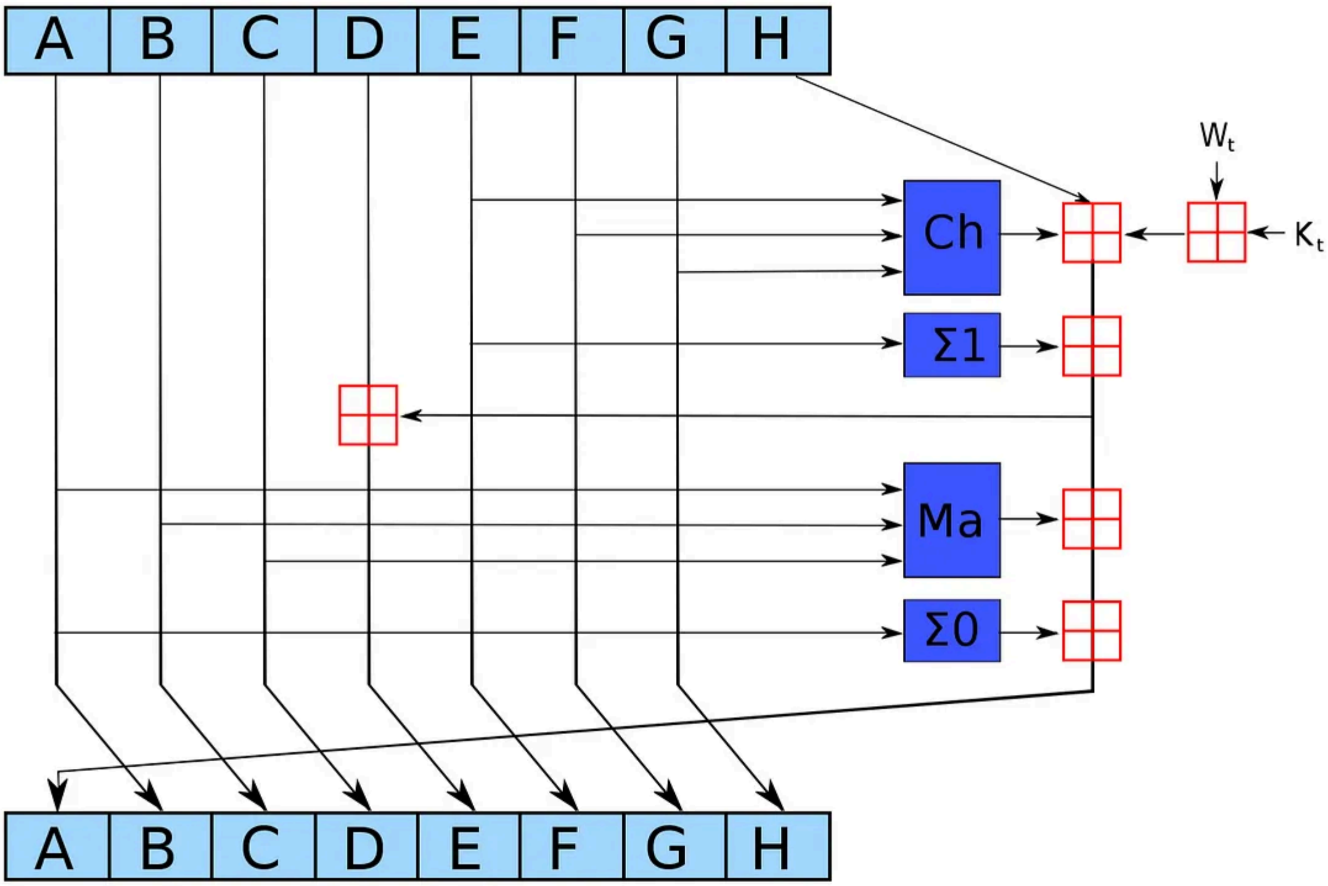
Block = M + 1 + 0$^{(P-1)}$ + L

Initializing Buffers

A_0 = 0x6a09e667
B_0 = 0xbb67ae85
C_0 = 0x3c6ef372
D_0 = 0xa54ff53a
E_0 = 0x510e527f
F_0 = 0x9b05688c
G_0 = 0x1f83d9ab
H_0 = 0x5be0cd19

$K_{[0..63]}$ = [0x428a2f98, 0x71374491, 0xb5c0fbcf, 0xe9b5dba5, 0x3956c25b, 0x59f111f1, 0x923f82a4, 0xab1c5ed5, 0xd807aa98, 0x12835b01, 0x243185be, 0x550c7dc3, 0x72be5d74, 0x80deb1fe, 0x9bdc06a7, 0xc19bf174,0xe49b69c1, 0xefbe4786, 0x0fc19dc6, 0x240ca1cc, 0x2de92c6f, 0x4a7484aa, 0x5cb0a9dc, 0x76f988da, 0x983e5152, 0xa831c66d, 0xb00327c8, 0xbf597fc7, 0xc6e00bf3, 0xd5a79147, 0x06ca6351, 0x14292967, 0x27b70a85, 0x2e1b2138, 0x4d2c6dfc, 0x53380d13, 0x650a7354, 0x766a0abb, 0x81c2c92e, 0x92722c85, 0xa2bfe8a1, 0xa81a664b, 0xc24b8b70, 0xc76c51a3, 0xd192e819, 0xd6990624, 0xf40e3585, 0x106aa070, 0x19a4c116, 0x1e376c08, 0x2748774c, 0x34b0bcb5, 0x391c0cb3, 0x4ed8aa4a, 0x5b9cca4f, 0x682e6ff3, 0x748f82ee, 0x78a5636f, 0x84c87814, 0x8cc70208, 0x90befffa, 0xa4506ceb, 0xbef9a3f7, 0xc67178f2]

Compression Round Operations



Chunk$_n$ = Block[512n, 512n + 512]
Chunk$_0$ = Block[0, 512]

C = Chunk
R = Operation Round

$$W(R) = \begin{cases} C[16R, 16R + 16], & R < 16 \\ W(R-16) + \sigma^0 + W(R-7) + \sigma^1, & R >= 16 \end{cases}$$

$$\sigma_0(R) = \texttt{ROTR}^7(W(R-15)) \ \oplus \ \texttt{ROTR}^{18}(W(R-15)) \oplus \ \texttt{SHR}^3(W(R-15))$$
$$\sigma_1(R) = \texttt{ROTR}^{17}(W(R-2)) \ \oplus \ \texttt{ROTR}^{19}(W(R-2)) \oplus \ \texttt{SHR}^{10}(W(R-2))$$

Simplified Equation

$$W(R) = \begin{cases} C[16R, 16R+16], \\ W(R-16) + (\texttt{ROTR}^{7}(W(R-15)) \ \oplus \ \texttt{ROTR}^{18}(W(R-15)) \oplus \ \texttt{SHR}^{3}(W(R-15))) + W(R-7) \end{cases}$$

ROTR^n(X) = Circular Right Rotation by n bits
(Right Shift n Bits) Bitwise OR (Left Shift by (Length$_X$ - n) Bits)
(X << n) OR (X << (Length$_X$ - n))

SHR^n(X) = Circular Right Shift by n bits
X >> n

Function Definitions

$$\begin{aligned} \texttt{Ch}_R &= (E_R \ \& \ F_R) \ \oplus \ (\tilde{E}_R \ \& \ G_R) \\ \Sigma 1_R &= (E_R \ggg 6) \ \oplus \ (E_R \ggg 11) \ \oplus \ (E_R \ggg 25) \\ \texttt{Ma}_R &= (A_R \ \& \ B_R) \ \oplus \ (A_R \ \& \ C_R) \ \oplus \ (B_R \ \& \ C_R) \\ \Sigma 0_R &= (A_R \ggg 2) \ \oplus \ (A_R \ggg 13) \ \oplus \ (A_R \ggg 22) \end{aligned}$$

Box Definitions

$$\begin{aligned} \texttt{Box}_1(R) &= (W_R + K_R) \bmod 2^{32} \\ \texttt{Box}_2(R) &= (\texttt{Ch}_R + H_R + \texttt{Box}_1(R)) \bmod 2^{32} \\ \texttt{Box}_3(R) &= (\texttt{Box}_2(R) + \Sigma 1_R) \bmod 2^{32} \\ \texttt{Box}_4(R) &= (D_R + \texttt{Box}_3(R)) \bmod 2^{32} \\ \texttt{Box}_5(R) &= (\texttt{Box}_3(R) + \texttt{Ma}_R) \bmod 2^{32} \\ \texttt{Box}_6(R) &= (\texttt{Box}_5(R) + \Sigma 0_R) \bmod 2^{32} \end{aligned}$$

Variable Definitions

$$\begin{aligned} A_{R+1} &= \texttt{Box}_6(R) \\ B_{R+1} &= A_R \\ C_{R+1} &= B_R \\ D_{R+1} &= C_R \\ E_{R+1} &= \texttt{Box}_5(R) \\ F_{R+1} &= E_R \\ G_{R+1} &= F_R \\ H_{R+1} &= G_R \end{aligned}$$

Simplified Equations

$$
\begin{aligned}
T_1 &= H_R + \Sigma 1_R + \mathtt{Ch}_R + K_R + W_R \\
T_2 &= \mathtt{Ma}_R + \Sigma 0_R \\
A_{R+1} &= T_1 + T_2 \\
B_{R+1} &= A_R \\
C_{R+1} &= B_R \\
D_{R+1} &= C_R \\
E_{R+1} &= D_R + T \\
F_{R+1} &= E_R \\
G_{R+1} &= F_R \\
H_{R+1} &= G_R
\end{aligned}
$$

Final Algorithm

$$
\begin{aligned}
T_1 &= H_R + ((E_R \ggg 6) \ \oplus \ (E_R \ggg 11) \ \oplus \ (E_R \ggg 25)) + ((E_R \,\&\, F_R) \ \oplus \ (\tilde{E}_R \,\&\, G_R)) + K_R + W_R \\
T_2 &= ((A_R \,\&\, B_R) \ \oplus \ (A_R \,\&\, C_R) \ \oplus \ (B_R \,\&\, C_R)) + ((A_R \ggg 2) \ \oplus \ (A_R \ggg 13) \ \oplus \ \oplus \ (A_R \ggg 22)) \\
A_{R+1} &= T_1 + T_2 \\
B_{R+1} &= A_R \\
C_{R+1} &= B_R \\
D_{R+1} &= C_R \\
E_{R+1} &= D_R + T \\
F_{R+1} &= E_R \\
G_{R+1} &= F_R \\
H_{R+1} &= G_R
\end{aligned}
$$

Pseudo-code

```
K = [0x428a2f98, 0x71374491, 0xb5c0fbcf, 0xe9b5dba5, 0x3956c25b, 0x59f111f1,
0x923f82a4, 0xab1c5ed5, 0xd807aa98, 0x12835b01, 0x243185be, 0x550c7dc3,
0x72be5d74, 0x80deb1fe, 0x9bdc06a7, 0xc19bf174,0xe49b69c1, 0xefbe4786,
0x0fc19dc6, 0x240ca1cc, 0x2de92c6f, 0x4a7484aa, 0x5cb0a9dc, 0x76f988da,
0x983e5152, 0xa831c66d, 0xb00327c8, 0xbf597fc7, 0xc6e00bf3, 0xd5a79147,
0x06ca6351, 0x14292967, 0x27b70a85, 0x2e1b2138, 0x4d2c6dfc, 0x53380d13,
0x650a7354, 0x766a0abb, 0x81c2c92e, 0x92722c85, 0xa2bfe8a1, 0xa81a664b,
0xc24b8b70, 0xc76c51a3, 0xd192e819, 0xd6990624, 0xf40e3585, 0x106aa070,
0x19a4c116, 0x1e376c08, 0x2748774c, 0x34b0bcb5, 0x391c0cb3, 0x4ed8aa4a,
0x5b9cca4f, 0x682e6ff3, 0x748f82ee, 0x78a5636f, 0x84c87814, 0x8cc70208,
0x90befffa, 0xa4506ceb, 0xbef9a3f7, 0xc67178f2]

function sha256(message)
    padding = "1"

    P = 512 * ceil((message.length + 1 + 64) / 512) - (message.length + 64)

    for i = 0 to P-1
```

```
        padding += "0"

    // 64 bit binary
    length = to_binary(message.length * 8, 64)
    block = message + padding + length

    {A = 0x6a09e667, B = 0xbb67ae85,
    C = 0x3c6ef372, D = 0xa54ff53a,
    E = 0x510e527f, F = 0x9b05688c,
    G = 0x1f83d9ab, H = 0x5be0cd19}

    chunks = []
    for i = 0 to (block.length // 512) - 1
        chunks[i] = block[512*i : 512*(i+1)

    for chunk of chunks
        words = []
        for i = 0 to 15
            words[i] = to_int(chunk[32*i : 32*(i+1)]

        {A, B, C, D, E, F, G, H} = compress(A, B, C, D, E, F, G, H,
W(words))

    hash = to_hex(A) + to_hex(B) + to_hex(C) + to_hex(D) + to_hex(E) +
to_hex(F) + to_hex(G) + to_hex(H) print("Output Hash - " + hash_output)
    print("Output Hash - " + hash)

function compress(A, B, C, D, E, F, G, H, W)
    A0 = A
    B0 = B
    C0 = C
    D0 = D
    E0 = E
    F0 = F
    G0 = G
    H0 = H

    for R = 0 to 63
        T1 = (H + (ROTR(6, E) ^ ROTR(11, E) ^ ROTR(25, E)) + ((E & F) ^ (~E
& G)) + K[R] + W[R]) & 0xFFFFFFFF
```

```
        T2 = (((A & B) ^ (A & C) ^ (B & C)) + (ROTR(2, A) ^ ROTR(13, A) ^ ROTR(22, A))) & 0xFFFFFFFF

        H = G
        G = F
        F = E
        E = (D + T1) & 0xFFFFFFFF
        D = C
        C = B
        B = A
        A = (T1 + T2) & 0xFFFFFFFF

    A = (A + A0) & 0xFFFFFFFF
    B = (B + B0) & 0xFFFFFFFF
    C = (C + C0) & 0xFFFFFFFF
    D = (D + D0) & 0xFFFFFFFF
    E = (E + E0) & 0xFFFFFFFF
    F = (F + F0) & 0xFFFFFFFF
    G = (G + G0) & 0xFFFFFFFF
    H = (H + H0) & 0xFFFFFFFF

    return {A, B, C, D, E, F, G, H}

function W(words)
    W = Array(64)

    for R = 0 to 15
        W[R] = words[R]

    for R = 16 to 63
        sigma0 = (ROTR(7, W[R-15]) ^ ROTR(18, W[R-15]) ^ SHR(3, W[R-15]))
        sigma1 = (ROTR(17, W[R-2]) ^ ROTR(19, W[R-2]) ^ SHR(10, W[R-2]))
        W[R] = (W[R-16] + sigma0 + W[R-7] + sigma1) & 0xFFFFFFFF

    return W

function ROTR(n, X)
    return ((X >> n) OR (X << (32 - n))) & 0xFFFFFFFF
```

```
function SHR(n, X)
    return (X >> n) & 0xFFFFFFFF
```