SHA256

1.0 Overview

SHA or Secure Hash Algorithm is used in several crypto protocols like DNSSEC. Before the message can be hashed it must be padded to have a length that is a multiple of 521 bits. Than the message is parsed int 521-bit message blocks $M^{(1)}, M^{(2)}, \dots, M^{(N)}$.

Beginning with the initial hash value $H^{(0)}$ the message blocks get sequentially computed.

$$H^{(i)} = H^{(i-1)} + C_{M^{(i)}}\left(H^{(i-1)}\right)$$

C is the SHA-256 compression function and + is mod 2^{64} addition. H(N) is the resulting hash of m.

1.1 Description of SHA-512

The SHA-256 compression function work with 512-bit message blocks and a 256-bit intermediate hash value. It is essentially a 256-bit block cipher algorithm which encrypts the intermediate hash value using the message block as key. Hence there are two main components to describe: (1) the SHA-256 compression function, and (2) the SHA-256 message schedule. The following notation gets used for further operation on 32-bit words

The following eight 32-bit words are the initial hash value:

$$H_1^{(0)} = \text{6a09e667}$$

$$H_2^{(0)} = \text{bb67ae85}$$

$$H_3^{(0)} = \text{3c6ef372}$$

$$H_4^{(0)} = \text{a54ff53a}$$

$$H_5^{(0)} = \text{510e527f}$$

$$H_6^{(0)} = \text{9b05688c}$$

$$H_7^{(0)} = \text{1f83d9ab}$$

$$H_8^{(0)} = \text{5be0cd19}$$

Pre-processing

1. Apply padding that the length of the message M in bits is l. Append a “1” bit at the end of the message, and then k zero bits. k is the smallest positive solution to the equation $l + 1 + k \equiv 448 \; mod \; 512$. Than append a 64-bit block which is equal to the number *l* in binary

2. When padding is done parse the message into N 512-bit blocks $M^{(1)}, M^{(2)}, \dots, M^{(N)}$. Every $M^{(x)}$ is a 32-bit block. To accomplish that the left-most is stored in the significant bit position the bi-endian convention is used.

Main loop

For $i = 1$ to N (N = number of blocks in the padded message)

{

/* Initialize registers $a;\ b;\ c;\ d;\ e;\ f;\ g;\ h$ with the $(i-1)^{st}$ intermediate hash value (= the initial hash value when $i = 1$) */

$a \leftarrow H_1^{(i-1)}$

$b \leftarrow H_2^{(i-1)}$

$\vdots$

$h \leftarrow H_8^{(i-1)}$

/* Apply the SHA-256 compression function to update registers $a; b; \dots; h$ */

For $j = 0$ to 63
{

Compute $Ch(e,f,g)$, $Maj(a,b,c)$, $\sum_0(a)$, $\sum_1(e)$, and Wj (see Definitions below)

$T_1 \leftarrow h + \sum_1(e) + Ch(e,f,g) + K_j + W_j$
$T_2 \leftarrow \sum_0(a) + Maj(a,b,c)$
$h \leftarrow g$
$g \leftarrow f$
$f \leftarrow e$
$e \leftarrow d + T_1$
$d \leftarrow c$
$c \leftarrow b$
$b \leftarrow a$
$a \leftarrow T_1 + T_2$
}

// Compute the i^{th} intermediate hash value $H^{(i)}$

$H_1^{(i)} \leftarrow a + H_1^{(i-1)}$
$H_2^{(i)} \leftarrow a + H_2^{(i-1)}$
$\vdots$
$H_8^{(i)} \leftarrow a + H_8^{(i-1)}$

}

$H^{(N)} = (H_1^{(N)}, H_2^{(N)}, \dots, H_8^{(N)})$ is the hash of M.

Definitions

Six logical functions are used in SHA-256. They are all computed with 32-bit words.

$$Ch(x, y, z) = (x \wedge \mathrm{y}) \oplus (\neg x \wedge \mathrm{z})$$
$$Maj(x, y, z) = (x \wedge \mathrm{y}) \oplus (x \wedge \mathrm{z}) \oplus (y \wedge \mathrm{z})$$
$$\sum_0(x) = S^2(x) \oplus S^{13}(x) \oplus S^{22}(x)$$
$$\sum_1(x) = S^2(x) \oplus S^{13}(x) \oplus S^{22}(x)$$
$$\sigma_0(x) = S^7(x) \oplus S^{18}(x) \oplus R^3(x)$$
$$\sigma_1(x) = S^{17}(x) \oplus S^{19}(x) \oplus R^{10}(x)$$

The SHA-256 message schedule is used to compute the Expanded message blocks $W_0, W_1, \ldots W_{63}$:

$W_j = M_j^{(i)}$ for $j = 0, 1, \ldots, 15$, and
For $j = 16$ to 63
{

$$W_j \leftarrow \sigma_1(W_{j-2}) + W_{j-7} + \sigma_0(W_{j-15}) + W_{j-16}$$

}

A sequence of constant words, $K_0, K_1, \ldots K_{63}$, is used in SHA-256.

428a2f98 71374491 b5c0fbcf e9b5dba5 3956c25b 59f111f1 923f82a4 ab1c5ed5 d807aa98 12835b01 243185be 550c7dc3 72be5d74 80deb1fe 9bdc06a7 c19bf174 e49b69c1 efbe4786 0fc19dc6 240ca1cc 2de92c6f 4a7484aa 5cb0a9dc 76f988da 983e5152 a831c66d b00327c8 bf597fc7 c6e00bf3 d5a79147 06ca6351 14292967 27b70a85 2e1b2138 4d2c6dfc 53380d13 650a7354 766a0abb 81c2c92e 92722c85 a2bfe8a1 a81a664b c24b8b70 c76c51a3 d192e819 d6990624 f40e3585 106aa070 19a4c116 1e376c08 2748774c 34b0bcb5 391c0cb3 4ed8aa4a 5b9cca4f 682e6ff3 748f82ee 78a5636f 84c87814 8cc70208 90befffa a4506ceb bef9a3f7 c67178f2

These are the first thirty-two bits of the fractional parts of the cube roots of the first sixty-four primes.

1.3 Diagrams

SHA-256 compression function:

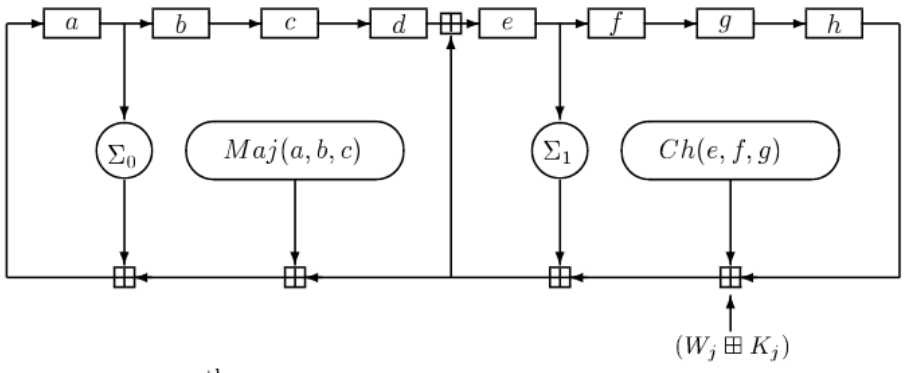


Figure 1: j^{th} internal step of the SHA-256 compression function C

Where the symbol $\boxplus$ denotes mod 2^{32} addition.

Message schedule:

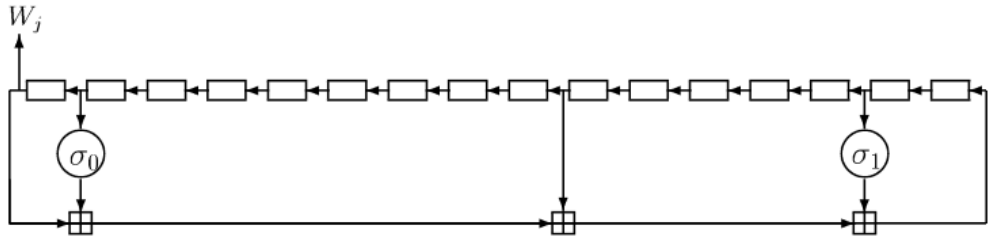


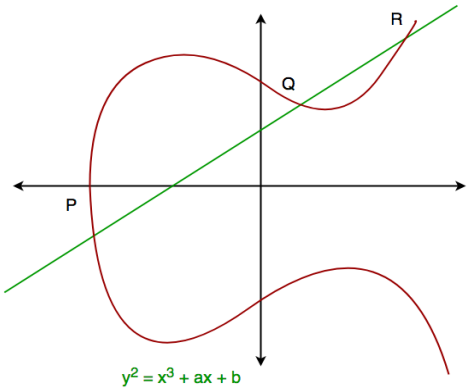
Figure 2: SHA-256 message schedule

The registers here are loaded with $W_0, W_1, \ldots W_{15}$.

Elliptic Curve Cryptography (ECC)

ECC is an approach to asymmetric key cryptography based on the structure on elliptic curves over a finite field. To understand Elliptic Curve Cryptography, it is important to understand the basics of elliptic curves. Elliptic curves are defined by the equation $y^2 = x^3 + ax + b$.

Elliptic curves could intersect in 3 point when a line is intersecting the curve. This intersection is used for encryption and decryption. ECC has a higher entropy than AES, for and 256-bit ECC key equivalent you need a 3072-bis RSA key.



This approach uses six tuple {P, a, b, G, n, h}

P = Field that the curve is define over

G = Generator point

a, b = Values define the curve

h = Co- factor

n = Prime order of G

Diffie-Hellman algorithm

The Diffie-Hellman algorithm is being used as and key distribution algorithm with various key asymmetric key algorithms like RSA. The Diffie-Hellman exchange algorithm is based on the discrete logarithm problem.

Client A	**Server B**
group generator g group size p	
private key α public key $a = g^{\alpha}$ mod p	private key β public key $b = g^{\beta}$mod p
Exchange of a and b	
Secret $s = b^{\alpha}$ mod p	Secret $s = a^{\alpha}$ mod p

AES

Advanced Encryption Standard is a part of the Rijndael block cipher developed by Vincent Rijmen and Joan Daemen.

The cypher works as a substitution-permutation network. AES has a fixed block size of 128 bits and key lengths of 128, 192 or 256.

Algorithm

The 128bit message block gets represented as a two-dimensional array:

$$\begin{bmatrix} b_0 & b_4 & b_8 & b_{12} \\ b_1 & b_5 & b_9 & b_{13} \\ b_2 & b_6 & b_{10} & b_{14} \\ b_3 & b_7 & b_{11} & b_{15} \end{bmatrix}$$

Depending on the Key size a different number of rounds get done.

Rounds	Key length
10	128-bit
12	192-bit
14	256-bit

Processing steps (1:1 von Wiki)

1. KeyExpansion-round keys re sutracted from the cipher key by using Rijndael's key schedule. AES neds for every round a 128-bit key block plus one more.
2. Initial round key addition:
    a. AddRoundKey—each block is combined with a block of the round key using bitwise xor.
3. n - 1 rounds
    a. SubBytes—is a non-linear substitution step where each byte is replaced with another according to a lookup table.
    b. ShiftRows—is a transposition step where the last three rows of the state are shifted cyclically a certain number of steps.
    c. MixColumns—a linear mixing operation which operates on the columns of the state, combining the four bytes in each column.
    d. AddRoundKey
4. Final round
    a. SubBytes
    b. ShiftRows
    c. AddRoundKey

<u>SubBytes</u>

During the sub-bytes step, each byte a in the block is replaced with a sub-byte $S(a)$ using an 8-bit substitution box. The S-box used is derived from the multiplicative inverse over GF(2^8).
$S(a_{i,j}) = b_{i,j}$

$$\begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix} \xrightarrow{SubBytes} \begin{bmatrix} b_{0,0} & b_{0,1} & b_{0,2} & b_{0,3} \\ b_{1,0} & b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,0} & b_{2,1} & b_{2,2} & b_{2,3} \\ b_{3,0} & b_{3,1} & b_{3,2} & b_{3,3} \end{bmatrix}$$

ShiftRows

During the Shift Rows step second row is shifted by one, the third by two and the third by 3 positions.

$$\begin{matrix} no\ change \\ shift1 \\ shift\ 2 \\ shift\ 3 \end{matrix} \begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix} \xrightarrow{ShiftRows} \begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,2} & a_{1,2} & a_{1,3} & a_{1,0} \\ a_{2,2} & a_{2,3} & a_{2,0} & a_{2,1} \\ a_{3,3} & a_{3,0} & a_{3,1} & a_{3,2} \end{bmatrix}$$

MixColumns

In the MixColumns step, the four bytes of each column of the state block is combined using an invertible linear transformation. During this operation, each column is transformed by multiplying with a fixed matrix:

$$\begin{bmatrix} b_{0,j} \\ b_{1,j} \\ b_{2,j} \\ b_{3,j} \end{bmatrix} = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \begin{bmatrix} a_{0,j} \\ a_{1,j} \\ a_{2,j} \\ a_{3,j} \end{bmatrix} \quad 0 \leq j \leq 3$$

$$\begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix} \xrightarrow{MixColumns} \begin{bmatrix} b_{0,0} & b_{0,1} & b_{0,2} & b_{0,3} \\ b_{1,0} & b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,0} & b_{2,1} & b_{2,2} & b_{2,3} \\ b_{3,0} & b_{3,1} & b_{3,2} & b_{3,3} \end{bmatrix}$$

AddRoundKey

During the AddRoundkey step a subkey fron the Rijndael's key schedule gets combined with the state block by using bitwise XOR:

$$\begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix} \begin{bmatrix} k_{0,0} & k_{0,1} & k_{0,2} & k_{0,3} \\ k_{1,0} & k_{1,1} & k_{1,2} & k_{1,3} \\ k_{2,0} & k_{2,1} & k_{2,2} & k_{2,3} \\ k_{3,0} & k_{3,1} & k_{3,2} & k_{3,3} \end{bmatrix} \xrightarrow{\text{XOR}} \begin{bmatrix} b_{0,0} & b_{0,1} & b_{0,2} & b_{0,3} \\ b_{1,0} & b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,0} & b_{2,1} & b_{2,2} & b_{2,3} \\ b_{3,0} & b_{3,1} & b_{3,2} & b_{3,3} \end{bmatrix}$$

Sources:

- https://en.wikipedia.org/wiki/SHA-2
- http://www.iwar.org.uk/comsec/resources/cipher/sha256-384-512.pdf
- https://www.geeksforgeeks.org/implementation-diffie-hellman-algorithm/
- https://www.korelstar.de/informatik/aes.html
- https://en.wikipedia.org/wiki/Advanced_Encryption_Standard#Description_of_the_ciphers