SHA-256: Secure Hashing Algorithm

Introduction

An n-bit *hash* is a map from arbitrary length messages to n-bit *hashvalues*. An n- bit cryptographic hash is an n-bit hash which is $one - way$ and $collision - resistant$. Such functions are important cryptographic primitives used for such things as digital signatures and password protection.SHA-256 is one of the most popular cryptographic hashing algorithms.

The SHA-256 compression function operates on a 512-bit message block 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.

Algorithm

This section shows the step-by-step methods taken by SHA-256 hash function.But before that here is a bunch of function notations and operations which are used in the computation.

Basic operations

- Boolean operations AND, XOR and OR, denoted by $\wedge$, $\oplus$ and $\vee$, respectively.
- Bitwise complement, denoted by $\neg$.
- Integer addition modulo 2^{32}, denoted by A + B.
- Each of them operates on 32-bit words. For the last operation, binary words are interpreted as integers written in base 2.
- $RotR(A, n)$ denotes the circular right shift of n bits of the binary word A.
- $ShR(A, n)$ denotes the right shift of n bits of the binary word A.
- $A||B$ denotes the concatenation of the binary words A and B.

Functions and Constants

The algorithm uses the functions:

$$\begin{aligned}
Ch(X,Y,Z) &= (X \wedge Y) \oplus (\neg X \wedge Z),\\
Maj(X,Y,Z) &= (X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z),\\
\Sigma_0(X) &= RotR(X,2) \oplus RotR(X,13) \oplus RotR(X,22),\\
\Sigma_1(X) &= RotR(X,6) \oplus RotR(X,11) \oplus RotR(X,25),\\
\sigma_0(X) &= RotR(X,7) \oplus RotR(X,18) \oplus ShR(X,3),\\
\sigma_1(X) &= RotR(X,17) \oplus RotR(X,19) \oplus ShR(X,10),
\end{aligned}$$

A sequence of constant words,$K_0, k_1, k_2, ..K_{63}$; is used in SHA-256. In hex, these are given by

```
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.

Pre-processing

Computation of hash message begins by preparing the message.

Padding

To ensure that the message has length multiple of 512 bits:

- first, a bit 1 is appended,

- next, k bits 0 are appended, with k being the smallest positive integer such that $l + 1 + k448 \ mod \ 512$, where l is the length in bits of the initial message,
- finally, the length$l < 2^{64}$ of the initial message is represented with exactly 64 bits, and these bits are added at the end of the message.

The message shall always be padded, even if the initial length is already a multiple of 512.

For example, the (8-bit ASCII) message ”abc” has length $8 * 3 = 24$so it is padded with a one, then $448 - (24 + 1) = 423$ zero bits, and then its length to become the 512-bit padded message

01100001 01100010 01100011 1|423 0's| its length in binary

Block decomposition

For each block $M \in 0, 1^{512}$, 64 words of 32 bits each are constructed as follows:

- the first 16 are obtained by splitting M in 32-bit blocks

$$M = W_1||W_2||W_3||...||W_{15}||W_{16}$$

- the remaining 48 are obtained with the formula:

$$W_i = \sigma_1(W_{i-2}) + W_{i-7} + \sigma_0(W_{i-15}) + W_{i-16} \qquad 17 \leq i \leq 64$$

Hash Computation

- First, eight variables are set to their initial values, given by the first 32 bits of the fractional part of the square roots of the first 8 prime numbers:

$H_1^{(0)}$ = `0x6a09e667`	$H_2^{(0)}$ = `0xbb67ae85`	$H_3^{(0)}$ = `0x3c6ef372`	$H_4^{(0)}$ = `0xa54ff53a`
$H_5^{(0)}$ = `0x510e527f`	$H_6^{(0)}$ = `0x9b05688c`	$H_7^{(0)}$ = `0x1f83d9ab`	$H_8^{(0)}$ = `0x5be0cd19`

- Next, the blocks $M^{(1)}, M^{(2)}, ...M^{(N)}$ are processed one at a time:

```
For $t = 1$ to $N$
{
    – construct the 64 blocks $W_i$ from $M^{(t)}$ , as explained in Block
      decomposition
    – set these eight values to
      $(a, b, c, d, e, f, g, h) = (H_1^{(t-1)}, H_2^{(t-1)}, H_3^{(t-1)}, H_4^{(t-1)}, H_5^{(t-1)}, H_6^{(t-1)}, H_7^{(t-1)}, H_8^{(t-1)})$
    – For $j = 1$ to 64
      {
      using the pre-computed data nad the functions defined in previous
      subsections
              $T_1 = h + \Sigma_1(e) + Ch(e, f, g) + K_j + W_j$
              $T_2 = \Sigma_0(a) + Maj(a, b, c)$
              $h = g$
              $g = f$
              $f = e$
              $e = d + T_1$
              $d = c$
              $c = b$
              $b = a$
              $a = T_1 + T_2$

      }
```

– compute the new value of $H_i^{(t)}$

$$\begin{aligned}
H_1^{(t)} &= H_1^{(t-1)} + a\\
H_2^{(t)} &= H_2^{(t-1)} + b\\
H_3^{(t)} &= H_3^{(t-1)} + c\\
H_4^{(t)} &= H_4^{(t-1)} + d\\
H_5^{(t)} &= H_5^{(t-1)} + e\\
H_6^{(t)} &= H_6^{(t-1)} + f\\
H_7^{(t)} &= H_7^{(t-1)} + g\\
H_8^{(t)} &= H_8^{(t-1)} + h
\end{aligned}$$

}

- The hash of the message is the concatenation of the variables H_i^N after the last block has been processed

$$H = H_1^{(N)}||H_2^{(N)}||H_3^{(N)}||H_4^{(N)}||H_5^{(N)}||H_6^{(N)}||H_7^{(N)}||H_8^{(N)}$$

Conclusion

If the above algorithm is implemented just as described and Defined the class SHA256 like:

public static Biginteger SHA256(M)

Input: M is a chain of bytes of arbitrary length;

Output: a positive integer in the interval $[0, 2^{256})$, the value of the hash of M.