**SHA ALGORITHM**

Suppose the message ‘abc’ were to be encoded using SHA-1, with the message ‘abc’ in binary being

01100001\ 01100010\ 01100011

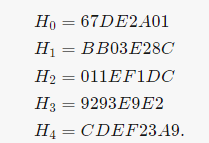
01100001 01100010 01100011

and that in hex being

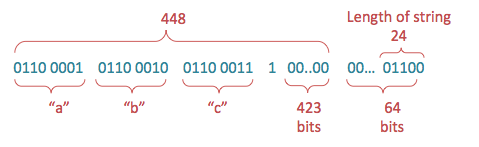
616263.

616263.

1) The first step is to initialize five random strings of hex characters that will serve as part of the hash function (shown in hex):

​

2) The message is then padded by appending a 1, followed by enough 0s until the message is 448 bits. The length of the message represented by 64 bits is then added to the end, producing a message that is 512 bits long:



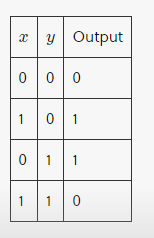
3) The padded input obtained above, M, is then divided into 512-bit chunks, and each chunk is further divided into sixteen 32-bit words, *W*0​…*W*15​. In the case of ‘abc’, there’s only one chunk, as the message is less than 512-bits total.

4) For each chunk, begin the 80 iterations, i, necessary for hashing (80 is the determined number for SHA-1), and execute the following steps on each chunk, Mn

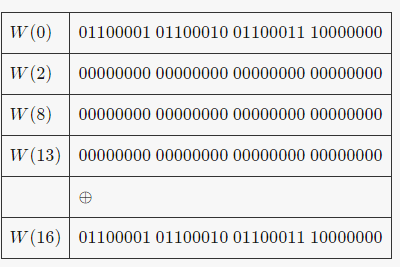
For iterations 16 through 79, where 16≤*i*≤79, perform the following operation:



Where XOR, or ⊕, is represented by the following comparison of inputs *x* and *y*:

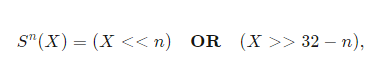


* For example, when *i* is 16, the words chosen are W(13), W(8), W(2), W(0) and the output is a new word, W(16) so performing the XOR, or ⊕, operation on those words will give this:

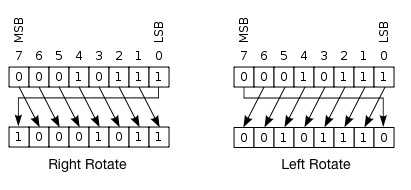


**Circular Shift Operation**

 The circular shift operation S^n(X)on the word *X* by *n* bits, *n* being an integer between 0 and 32, is defined by



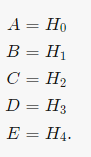
Where *X*<<*n* is the **left-shift** operation, obtained by discarding the leftmost *n* bits of *X* and padding the result with *n* zeroes on the right



So, a left shift S^n(W(i)) where *W*(*i*) is 10010, would produce 01001, as the rightmost bit 0 is shifted to the left side of the string. Therefore, *W*(16) would end up being

11000010 11000100 11000111 000000000

**5)** Now, store the hash values defined in step 1 in the following variables:

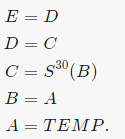


**6)** For 80 iterations, where 0≤*i*≤79, compute

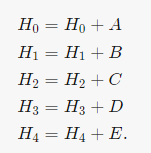


Details on the logical function, *f*, and on the values of *K(i).*

Reassign the following variables:



**7)** Store the result of the chunk’s hash to the overall hash value of all chunks, as shown below, and proceed to execute the next chunk:

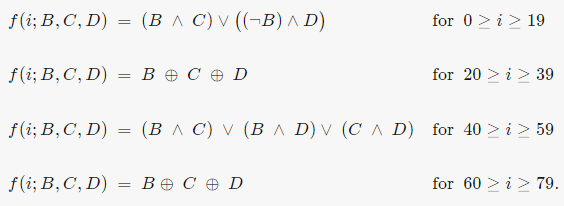


**8)** As a final step, when all the chunks have been processed, the message digest is represented as the 160-bit string comprised of the **OR** logical operator, ∨, of the 5 hashed values:



**Functions used in the algorithm**

A sequence of logical functions are used in SHA-1, depending on the value of *i*, where 0≤*i*≤79, and on three 32-bit words B, C, and D, in order to produce a 32-bit output. The following equations describe the logical functions, where ¬ is the logical NOT, ∨ is the logical OR, ∧ is the logical AND, and ⊕ is the logical XOR:



Additionally, a sequence of constant words, shown in hex below, is used in the formulas:

