Network Working Group
Request for Comments: 3174
Category: Informational

D. Eastlake, 3rd
Motorola
P. Jones

Example:

```
        01101100101110011101001001111011
XOR     01100101110000010110100110110111
        --------------------------------
  =     00001001011110001011101111001100
```

b. The operation X + Y is defined as follows: words X and Y represent integers x and y, where 0 <= x < 2^32 and 0 <= y < 2^32. For positive integers n and m, let n mod m be the remainder upon dividing n by m. Compute

   z = (x + y) mod 2^32.

   Then 0 <= z < 2^32. Convert z to a word, Z, and define Z = X + Y.

c. The circular left shift operation S^n(X), where X is a word and n is an integer with 0 <= n < 32, is defined by

   S^n(X) = (X << n) OR (X >> 32-n).

   In the above, X << n is obtained as follows: discard the left-most n bits of X and then pad the result with n zeroes on the right (the result will still be 32 bits). X >> n is obtained by discarding the right-most n bits of X and then padding the result with n zeroes on the left. Thus S^n(X) is equivalent to a circular shift of X by n positions to the left.

4. Message Padding

SHA-1 is used to compute a message digest for a message or data file that is provided as input. The message or data file should be considered to be a bit string. The lenghe d Ithe message is the number d Ibits inIthe message (the empty message has lenghe 0). If the number d Ibits inIa message is a multiple d I8, for compactness we can represent the message in hex. The purpose d Imessage padding is to make the total lenghe d Ia padded message a multiple d I512. SHA-1 sequentially processes blocks d I512Ibits when computing the message digest. The following specifies how this padding shall be performed. As a summary, a "1" followed by m "0"s followed by a 64-bit integer are appended to the end d Ithe message to produce a

5. Functions and Constants Used

Before processing any blocks, the H's are initialized as follows: in hex,

H0 = 67452301

H1 = EFCDAB89

H2 = 98BADCFE

H3 = 10325476

H4 = C3D2E1F0.

Now M(1), M(2), ... , M(n) are processed. To process M(i), we proceed as follows:

a. Divide M(i) into 16 words W(0), W(1), ... , W(15), where W(0) is the left-most word.

b. For t = 16 to 79 let

W(t) = S^1(W(t-3) XOR W(t-8) XOR W(t-14) XOR W(t-16)).

c. Let A = H0, B = H1, C = H2, D = H3, E = H4.

d. For t = 0 to 79 do

TEMP = S^5(A) + f(t;B,C,D) + E + W(t) + K(t);

E = D; D = C; C = S^30(B); B = A; A = TEMP;

e. Let H0 = H0 + 4 Td B = = (S^6mSj C, H3 H+ D, H4 H4))Tj 0 -11 Td(

Aftmbee processin, M(nks, thmoceage digemos is th160-bit strssi)

= H = H= H4.

T thmothod aboveed sumocis atis thsequenceds W(0), ... , W79(i)s))Tj 0 -11 Td(implement

```
int SHA1Reset(  SHA1Context *);
int SHA1Input(  SHA1Context *,
                const uint8_t *,
                unsigned int);
int SHA1Result( SHA1Context *,
                uint8_t Message_Digest[SHA1HashSize]);

#endif
```

7.2 .c file

```
/*
 *  sha1.c
 *
 *  Description:
```

```
#include "sha1.h"

/*
 *  Define the SHA1 circular left shift macro
 */
#define SHA1CircularShift(bits,word) \
```

```
    return shaSuccess;
```

```
 *  Description:
 *      According to the standard, the message must be padded to an even
 *      512 bits.  The first padding bit must be a '1'.  The last 64
 *      bits represent the length of the original message.  All bits in
 *      between should be 0.  This function will pad the message
 *      according to those rules by filling the Message_Block array
 *      accordingly.  It will also call the ProcessMessageBlock function
 *      provided appropriately.  When it returns, it can be assumed that
 *      the message digest has been computed.
 *
```

```
            context->Message_Block[context->Message_Block_Index++] = 0;
        }
    }

    /*
     *  Store the message length as the last 8 octets
     */
    context->Message_Block[56] = context->Length_High >> 24;
    context->Message_Block[57] = context->Length_High >> 16;
    context->Message_Block[58] = context->Length_High >> 8;
    context->Message_Block[59] = context->Length_High;
    context->Message_Block[60] = context->Length_Low >> 24;
    context->Message_Block[61] = context->Length_Low >> 16;
    context->Message_Block[62] = context->Length_Low >> 8;
    context->Message_Block[63] = context->Length_Low;

    SHA1ProcessMessageBlock(context);
}
```

7.3 Test Driver

The following code is a main program test driver to exercise the code in sha1.c.

```
/*
 *  sha1test.c
 *
 *  Description:
 *      This file will exercise the SHA-1 code performing the three
 *      tests documented in FIPS PUB 180-1 plus one which calls
 *      SHA1Input with an exact multiple of 512 bits, plus a few
 *      error test checks.
 *
 *  Portability Issues:
 *      None.
 *
 */

#include <stdint.h>
#include <stdio.h>
#include <string.h>
#include "sha1.h"

/*
 *  Define patterns for testing
 */
#define TEST1   "abc"
#define TEST2a  "abcdbcdecdefdefgefghfghighijhi"
```

```
            err = SHA1Input(&sha,
                  (const unsigned char *) testarray[j],
                  strlen(testarray[j]));
            if (err)
            {
                fprintf(stderr, "SHA1Input Error %d.\n", err );
                break;    /* out of for i loop */
            }
        }

        err = SHA1Result(&sha, Message_Digest);
        if (err)
        {
            fprintf(stderr,
            "SHA1Result Error %d, could not compute message digest.\n",
            err );
        }
        else
        {
            printf("\t");
            for(i = 0; i < 20 ; ++i)
            {
                printf("%02X ", Message_Digest[i]);
            }
            printf("\n");
        }
        printf("Should match:\n");
        printf("\t%s\n", resultarray[j]);
    }

    /* Test some error returns */
    err = SHA1Input(&sha,(const unsigned char *) testarray[1], 1);
    printf ("\nError %d. Should be %d.\n", err, shaStateError );
    err = SHA1Reset(0);
    printf ("\nError %d. Should be %d.\n", err, shaNull );
    return 0;
}
```

8. Security Considerations

Full Copyright Statement