Network Working Group P. Cheng
Request for Comments: 2202 IBM
Category: Informational R. Glenn

NIST

September 1997

Test Cases for HMAC-MD5 and HMAC-SHA-1

Status of This Memo

Abstract

This document provides two sets of test cases for HMAC-MD5 and HMAC-SHA-1, respectively. HMAC-MD5 and HMAC-SHA-1 are two constructs of the HMAC [HMAC] message authentication function using the MD5 [MD5] hash function and the SHA-1 [SHA] hash function. Both constructs are used by IPSEC [OG,CG] and other protocols to authenticate messages. The test cases and results provided in this document are meant to be used as a conformance test for HMAC-MD5 and HMAC-SHA-1 implementations.

1. Introduction

The general method for constructing a HMAC message authentication function using a particular hash function is described in section 2 of [HMAC]. We will not repeat the description here. Section 5 of [HMAC] also discusses truncating the output of HMAC; the rule is that we should keep the more significant bits (the bits in the left, assuming a network byte order (big-endian)).

In sections 2 and 3 we provide test cases for HMAC-MD5 and HMAC-SHA-1, respectively. Each case includes the key, the data, and the result. The values of keys and data are either hexadecimal numbers (prefixed by "0x") or ASCII character strings in double quotes. If a

value is an ASCII character string, then the HMAC computation for the corresponding test case DOES NOT include the trailing null character ('\0') in the string.

The C source code of the functions used to generate HMAC-SHA-1 results is listed in the Appendix. Note that these functions are meant to be simple and easy to understand; they are not optimized in any way. The C source code for computing HMAC-MD5 can be found in [MD5]; or you can do a simple modification to HMAC-SHA-1 code to get HMAC-MD5 code, as explained in the Appendix.

The test cases in this document are cross-verified by three independent implementations, one from NIST and two from IBM Research. One IBM implementation uses optimized code that is very different from the code in the Appendix. An implemenation that concurs with the results provided in this document should be interoperable with other similar implemenations. We do not claim that such an implementation is absolutely correct with respect to the HMAC definition in [HMAC].

2. Test Cases for HMAC-MD5

```
test_case =     1
key =           0x0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b
key_len =       16
data =          "Hi There"
data_len =      8
digest =        0x9294727a3638bb1c13f48ef8158bfc9d

test_case =     2
key =           "Jefe"
key_len =       4
data =          "what do ya want for nothing?"
data_len =      28
digest =        0x750c783e6ab0b503eaa86e310a5db738

test_case =     3
key =           0xaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

```
key_len         16
data =          0xdd repeated 50 times
data_len =      50
digest =        0x56be34521d144c88dbb8c733f0e8b3f6

test_case =     4
key =           0x0102030405060708090a0b0c0d0e0f10111213141516171819
key_len         25
data =          0xcd repeated 50 times
data_len =      50
digest =        0x697eaf0aca3a3aea3a75164746ffaa79
```



```
test_case =     5
key =           0x0c0c0c0c0c0c0c0c0c0c0c0c0c0c0c0c
key_len =       16
data =          "Test With Truncation"
data_len =      20
digest =        0x56461ef2342edc00f9bab995690efd4c
digest-96       0x56461ef2342edc00f9bab995

test_case =     6
key =           0xaa repeated 80 times
key_len =       80
data =          "Test Using Larger Than Block-Size Key - Hash Key
First"
data_len =      54
digest =        0x6b1ab7fe4bd7bf8f0b62e6ce61b9d0cd

test_case =     7
key =           0xaa repeated 80 times
key_len =       80
data =          "Test Using Larger Than Block-Size Key and Larger
                Than One Block-Size Data"
data_len =      73
digest =        0x6f630fad67cda0ee1fb1f562db3aa53e
```

3. Test Cases for HMAC-SHA-1

```
test_case =     1
key =           0x0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b
key_len =       20
data =          "Hi There"
data_len =      8
digest =        0xb617318655057264e28bc0b6fb378c8ef146be00

test_case =     2
key =           "Jefe"
key_len =       4
```

```
data =          "what do ya want for nothing?"
data_len =      28
digest =        0xeffcdf6ae5eb2fa2d27416d5f184df9c259a7c79

test_case =     3
key =           0xaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
key_len =       20
data =          0xdd repeated 50 times
data_len =      50
digest =        0x125d7342b9ac11cd91a39af48aa17b4f63f175d3
```



```
test_case =     4
key =           0x0102030405060708090a0b0c0d0e0f10111213141516171819
key_len =       25
data =          0xcd repeated 50 times
data_len =      50
digest =        0x4c9007f4026250c6bc8414f9bf50c86c2d7235da

test_case =     5
key =           0x0c0c0c0c0c0c0c0c0c0c0c0c0c0c0c0c0c0c0c0c
key_len =       20
data =          "Test With Truncation"
data_len =      20
digest =        0x4c1a03424b55e07fe7f27be1d58bb9324a9a5a04
digest-96 =     0x4c1a03424b55e07fe7f27be1

test_case =     6
key =           0xaa repeated 80 times
key_len =       80
data =          "Test Using Larger Than Block-Size Key - Hash Key
First"
data_len =      54
digest =        0xaa4ae5e15272d00e95705637ce8a3b55ed402112

test_case =     7
key =           0xaa repeated 80 times
key_len =       80
data =          "Test Using Larger Than Block-Size Key and Larger
                Than One Block-Size Data"
data_len =      73
digest =        0xe8e99d0f45237d786d6bbaa7965c7808bbff1a91
data_len =      20
digest =        0x4c1a03424b55e07fe7f27be1d58bb9324a9a5a04
digest-96 =     0x4c1a03424b55e07fe7f27be1

test_case =     6
key =           0xaa repeated 80 times
key_len =       80
data =          "Test Using Larger Than Block-Size Key - Hash Key
```

```
First"
data_len =      54
digest =        0xaa4ae5e15272d00e95705637ce8a3b55ed402112

test_case =     7
key =           0xaa repeated 80 times
key_len =       80
data =          "Test Using Larger Than Block-Size Key and Larger
                Than One Block-Size Data"
data_len =      73
digest =        0xe8e99d0f45237d786d6bbaa7965c7808bbff1a91
```


4. Security Considerations

This docuemnt raises no security issues. Discussion on the strength of the HMAC construction can be found in [HMAC].

References

[HMAC] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 1997.

[MD5] Rivest, R., "The MD5 Message-Digest Algorithm", RFC 1321, April 1992.

[SHA] NIST, FIPS PUB 180-1: Secure Hash Standard, April 1995.

[OG] Oehler, M., and R. Glenn, "HMAC-MD5 IP Authentication with Replay Prevention", RFC 2085, February 1997.

[CG] Chang, S., and R. Glenn, "HMAC-SHA IP Authentication with Replay Prevention", Work in Progress.

Authors' Addresses

Pau-Chen Cheng
IBM T.J. Watson Research Center
P.O.Box 704
Yorktown Heights, NY 10598

EMail: pau@watson.ibm.com

Robert Glenn
NIST
Building 820, Room 455
Gaithersburg, MD 20899

EMail: rob.glenn@nist.gov

Appendix

This appendix contains the C reference code which implements HMAC-SHA-1 using an existing SHA-1 library. It assumes that the SHA-1 library has similar API's as those of the MD5 code described in RFC 1321. The code for HMAC-MD5 is similar, just replace the strings "SHA" and "sha" with "MD5" and "md5". HMAC-MD5 code is also listed in RFC 2104.

```
#ifndef SHA_DIGESTSIZE
#define SHA_DIGESTSIZE  20
#endif

#ifndef SHA_BLOCKSIZE
#define SHA_BLOCKSIZE   64
#endif

#ifndef MD5_DIGESTSIZE
#define MD5_DIGESTSIZE  16
#endif

#ifndef MD5_BLOCKSIZE
#define MD5_BLOCKSIZE   64
#endif

/* Function to print the digest */
void
pr_sha(FILE* fp, char* s, int t)
{
        int     i ;

        fprintf(fp, "0x") ;
        for (i = 0 ; i < t ; i++)
                fprintf(fp, "%02x", s[i]) ;
        fprintf(fp, "0) ;
}

void truncate
(
 char*   d1,   /* data to be truncated */
 char*   d2,   /* truncated data */
 int     len   /* length in bytes to keep */
```

```
)
{
        int     i ;
        for (i = 0 ; i < len ; i++) d2[i] = d1[i];
}
```



```
/* Function to compute the digest */
void
hmac_sha
(
 char*    k,     /* secret key */
 int      lk,    /* length of the key in bytes */
 char*    d,     /* data */
 int      ld,    /* length of data in bytes */
 char*    out,   /* output buffer, at least "t" bytes */
 int      t
)
{
        SHA_CTX ictx, octx ;
        char    isha[SHA_DIGESTSIZE], osha[SHA_DIGESTSIZE] ;
        char    key[SHA_DIGESTSIZE] ;
        char    buf[SHA_BLOCKSIZE] ;
        int     i ;

        if (lk > SHA_BLOCKSIZE) {

                SHA_CTX         tctx ;

                SHAInit(&tctx) ;
                SHAUpdate(&tctx, k, lk) ;
                SHAFinal(key, &tctx) ;

                k = key ;
                lk = SHA_DIGESTSIZE ;
        }

        /**** Inner Digest ****/

        SHAInit(&ictx) ;

        /* Pad the key for inner digest */
        for (i = 0 ; i < lk ; ++i) buf[i] = k[i] ^ 0x36 ;
        for (i = lk ; i < SHA_BLOCKSIZE ; ++i) buf[i] = 0x36 ;

        SHAUpdate(&ictx, buf, SHA_BLOCKSIZE) ;
        SHAUpdate(&ictx, d, ld) ;

        SHAFinal(isha, &ictx) ;

        /**** Outter Digest ****/
```

```
            SHAInit(&octx) ;

            /* Pad the key for outter digest */
```



```
            for (i = 0 ; i < lk ; ++i) buf[i] = k[i] ^ 0x5C ;
            for (i = lk ; i < SHA_BLOCKSIZE ; ++i) buf[i] = 0x5C ;

            SHAUpdate(&octx, buf, SHA_BLOCKSIZE) ;
            SHAUpdate(&octx, isha, SHA_DIGESTSIZE) ;

            SHAFinal(osha, &octx) ;

            /* truncate and print the results */
            t = t > SHA_DIGESTSIZE ? SHA_DIGESTSIZE : t ;
            truncate(osha, out, t) ;
            pr_sha(stdout, out, t) ;

    }
```