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

Status of This Memo

This memo provides information for the Internet community. This memo does not specify an Internet standard of any kind. Distribution of this memo is unlimited.

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 implementation that concurs with the results provided in this document should be interoperable with other similar implementations. 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

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Key</th>
<th>Key Length</th>
<th>Data</th>
<th>Data Length</th>
<th>Digest</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0x0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b</td>
<td>16</td>
<td>&quot;Hi There&quot;</td>
<td>8</td>
<td>0x9294727a3638bb1c13f48ef8158bfc9d</td>
</tr>
<tr>
<td>2</td>
<td>&quot;Jefe&quot;</td>
<td>4</td>
<td>&quot;what do ya want for nothing?&quot;</td>
<td>28</td>
<td>0x750c783e6ab0b503eaa86e310a5db738</td>
</tr>
<tr>
<td>3</td>
<td>0xaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa</td>
<td>16</td>
<td>0xdd repeated 50 times</td>
<td>50</td>
<td>0x56be34521d144c88d8b8c733f0e8b3f6</td>
</tr>
<tr>
<td>4</td>
<td>0x0102030405060708090a0b0c0d0e0f10111213141516171819</td>
<td>25</td>
<td>0xcd repeated 50 times</td>
<td>50</td>
<td>0x697eaf0aca3a3aeaa3a75164746ffaa79</td>
</tr>
</tbody>
</table>
3. Test Cases for HMAC-SHA-1

test_case = 1
key = 0x0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0
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 = 0xefc6f6ae5eb2fa2d27416d5f184df9c259a7c79

test_case = 3
key = 0xaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
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 = 0xc1a03424b55e07fe7f27be1d58bb9324a9a5a04
digest-96 = 0xc1a03424b55e07fe7f27be1

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 = 0xc1a03424b55e07fe7f27be1d58bb9324a9a5a04
digest-96 = 0xc1a03424b55e07fe7f27be1

Cheng & Glenn                Informational                      [Page 4]
4. Security Considerations

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

References


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.

```c
#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, "0x") ;
}

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) ;

    /**** Outer Digest ****/
    SHAInit(&octx) ;

    /* Pad the key for outer 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);