Advancing the Cryptographic Hash-Based Message Authentication Code

Manish Kumar, Ashish Avasthi, and Gaurav Mishra

***Abstract*—In today's word the authentication is very much required for the purpose of the transfer of the information from one place to another. Now a day's the use of E-mail become very popular because of their fast and easy to use in nature. This new technology has led to the problem of Authentication of the Message. To perform this activity we use the technique called MAC (Message Authentication Code). The message authentication code is the digest that is send along with the message to authenticate the origin of the message from where it is generated. The MAC (Message Authentication Code) is generated by the process of creating the message digest and also adding the encryption to it. However this is not very secure. In this paper we will encrypt the message digest and then again use the previously available cryptographic algorithm and again encrypt the Message. We this the idea of confusion and diffusion make are message more secure.**

***Index Term*—Authentication, Digest, Encryption, Secure, Cryptographic Solution.**

I. Introduction

The Concept of Message Authentication Code (MAC) is quite similar to the message digest. The message digest is a finger print or the summary of the message. It is similar to the concepts Longitudinal Redundancy Check (LRC) or Cyclic Redundancy Check (CRC). That is, it is used to verify the integrity of the data (i.e. to ensure that a message has not been tampered with after it leaves the sender but before it reaches the receiver). Suppose a block of bits is organized in the form of a list (as rows) in the Longitudinal Redundancy Check (LRC). Here for instance, if we want to send 32 bits, we bits, we arrange them into a list of four (horizontals) rows. Then we count how many single bits occur in each of 8(vertical) columns.[if number of 1s in the columns is odd, then we say that the column has odd parity (indicated by a 1 bit in the shaded LRC rows); otherwise if the number of 1s in column is parity(indicated by a 0 bit in the shaded LRC).] for instance , in the first column, we have two 1s, indicating an even parity and have three 1s , indicating an odd parity, therefore we have a 1 in shaded LRC row for the last column. Thus, the parity for each column is calculated and a new row of eight parity bits is created these becomes the parity bits for the whole block.

Manuscript received November11, 2010; revised April 13, 2011.

Manish Kumar, Uttar Pradesh Technical University, CSE Dept., SRMSIBS, Member IACSIT, Lucknow, Uttar Pradesh, India, E-mail: dr.manish.2000@gmail.com

Ashish Avasthi, Uttar Pradesh Technical University, MCA Dept., SRMCEM. Member IACSIT, Lucknow, Uttar Pradesh, India, E-mail: ashishsrmcem@gmail.com

Gaurav Mishra, Uttar Pradesh Technical University, MCA Dept., SRMCEM. Member IACSIT, Lucknow, Uttar Pradesh, India, E-mail: gauravhanu@rediffmail.com

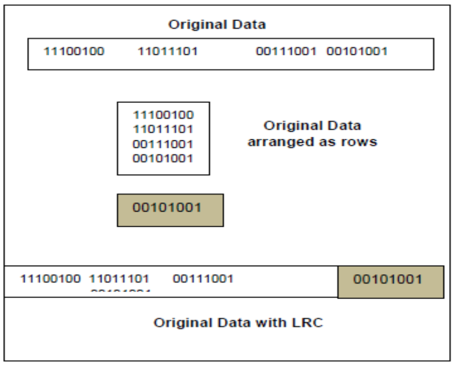


Figure 1

Thus the LRC is actually a fingerprint of the original message. The concept of message digests is based on similar principles. However, it is slightly wider in scope. For instance, suppose we have a number 4000 and we divide it by 4 to get 1000, 4 became a fingerprint of number 4000. Dividing 4000 by4 will always yield 1000. If we change either 4000 or 4, the result will not be 1000.

A. *Message Authentication Code (MAC)*

The concept of Message Authentication code is quite similar to that of a message digest. However, there is one difference. As we have seen, a message digest is simply a fingerprint of a message. There is no cryptography process involves in the case of message digests. In contrast, a MAC requires that the sender and receiver should know a shared symmetric (secret) key, which is used in the preparation of the MAC.

Thus, MAC involves cryptography processing. Let us see how this works.

Let us assume that the sender A wants to send a message M to received B.

1. A and B share a Symmetric(secret) key k,which is not known to anyone else A calculates the MAC by applying key k to message m.

2. A then sends the original message M and the MAC H1 to B.

3. When B received the message, B also used K to calculate its own MAC H2 over M.

4. B now compares H1 with H2. If the two match, B concludes that the message M has not been changed during transits. However, if H1 # H2, B rejects the message, realizing that the message was changed during transits.

The significations of a MAC are as Follows:

1. The MAC assures the receiver (in this case B) that the message is not altered. This is become an attacker alters the message but does not alter the MAC (in this case, H1), then the receiver's calculation of the MAC (in this case, H2) will differ from it. As we know key used the calculation of the MAC (in this case k) is assumed to be known only to the sender and receiver (in this case, A and B). Therefore, the

attacker does not know the key, K and therefore, she cannot alter the MAC.

2. The receiver (in this case B) is assumed that the message indeed came from the correct sender(in this case, B).since only the sender and receiver (A and B, respectively, in this case) know the secret key (in this case, K), no one else could have calculated the MAC (in this case H1) sent by the sender (in this case A).

II. Proposed Model

In our model we use the Hash based Message authentication model. The fundamental idea behind HMAC is to reuse the existing message digest algorithm like MD5 or SHA-1. In our proposed model the message entered is first encrypted by using the DES Algorithm which produces the cipher text now this cipher text that is produced its message digest is used and send it to the other location using the symmetric key encryption the other location receiver now reverse the process and compare the message authentication code of the message if both MAC founds to be same then no alteration of the message is done during the transmission. This produced MAC is more secure as it is encrypted by the DES encrypted. We can also use any other cryptographic algorithm in place of the DES Algorithm in our proposed model.

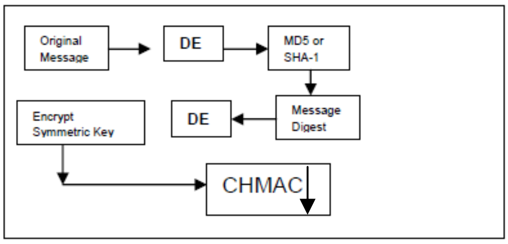


Figure 2

Working

The working of our proposed model consists of the following abbreviations:-

MD = The Message Digest/Hash function used
M = The input message whose MAC is to be calculate
M1 = Original Message whose encrypted text has to be calculated and denoted as the message whose digest has to be calculated.
K1 = First DES Key
K2 = Second DES Key
L = The number of blocks in the message M
b = The number of bits in each block
K = The shared symmetric key to be used in CHMAC
ipad =A string 00110110 repeated b/8 times opad= A string 01011010 repeated b/8 times Now with these variables we will discuss the different steps:

Step 1. Make the length of K equal to b. The algorithm starts with three possibilities, depending on the length of the key K :

• Length of K<b

In this case we will expand the key (K) to make the length of K equal to the number of bits in the original message block (b). for this we add as many 0 bits as required to the left of K.

• Length of K=b (in this case do nothing)

• Length of K>b

In this we need to trim K to make the length of K equal to the number of bits in the original message block (b) . for this we pass K through the message digest algorithm (H).

Step 2. XOR K with ipad to produce S1

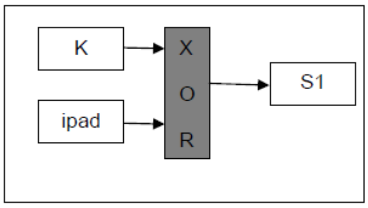


Figure 3

Step 3. Encrypt the original message M1 with DES and key (K1) now the output produce is again encrypted using DES and key (K2) and now the output of this is called as Original message (M). Step 4 Append M to S1. Now we take the original message (M) and simply append it to the end of S1.

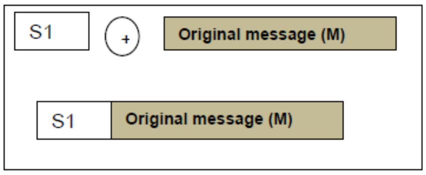


Figure 4

Step 5. Message Digest algorithm which has been selected is applied to the output of the step 4. And the output of this step is called as H.

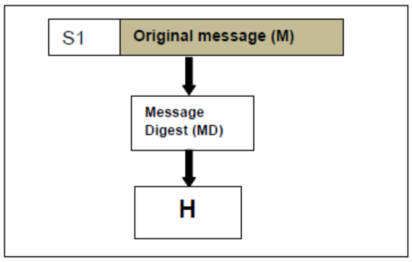
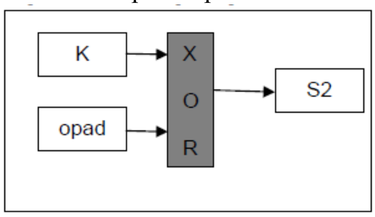


Figure 5

Step6 . XOR R with opad to produce S2



Step 7. Append H to S2. In this step, we take the message digest calculated in the step 5 and simply append it to the end of s2.

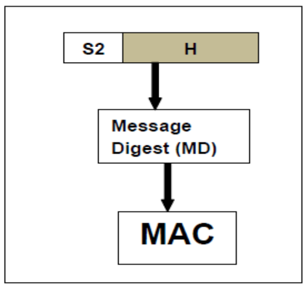


Figure 8

Step 8**. Message Digest Algorithm**. Now the selected message digest algorithm is applied to the output of step 7. This is the final message authentication code (MAC) that we want.

III. Implementation

The HMAC algorithm is specified for an arbitrary Approved cryptographic hash function, H. With minor modifications, an HMAC implementation can easily replace one hash function, H, with another hash function, H'. Conceptually, the intermediate results of the compression function on the B-byte blocks (K0 ⊕ ipad) and (K0 ⊕ opad) can be pre computed once, at the time of generation of the key K, or before its first use. These intermediate results can be stored and then used to initialize H each time that a message needs to be authenticated using the same key. For each authenticated message using the key K, this method saves the application of the hash function of H on two B-byte blocks (i.e., on (K ⊕ ipad) and (K ⊕ opad)). This saving may be significant when authenticating short streams of data. These stored intermediate values shall be treated and protected in the same manner as secret keys. The practical implementation is also shown on the various results by using the example of the HMAC and SHA-1 Digest Algorithm and also by introducing the double DES cryptographic algorithm.

HMAC EXAMPLES

These examples are provided in order to promote correct implementations of HMAC. The SHA-1 hash function used in these examples is specified in [4].
A.1 SHA-1 with 64-Byte Key
Text: "Sample #1"

Key: 00010203 04050607 08090a0b 0c0d0e0f
10111213 14151617 18191a1b 1c1d1e1f
20212223 24252627 28292a2b 2c2d2e2f
30313233 34353637 38393a3b 3c3d3e3f

K0: 00010203 04050607 08090a0b 0c0d0e0f
10111213 14151617 18191a1b 1c1d1e1f
20212223 24252627 28292a2b 2c2d2e2f
30313233 34353637 38393a3b 3c3d3e3f

K0 ⊕ ipad:

36373435 32333031 3e3f3c3d 3a3b3839
26272425 22232021 2e2f2c2d 2a2b2829
16171415 12131011 1e1f1c1d 1a1b1819
06070405 02030001 0e0f0c0d 0a0b0809

(Key ⊕ ipad)||text:

36373435 32333031 3e3f3c3d 3a3b3839
26272425 22232021 2e2f2c2d 2a2b2829
16171415 12131011 1e1f1c1d 1a1b1819
06070405 02030001 0e0f0c0d 0a0b0809
53616d70 6c652023 31

Hash((Key ⊕ ipad)||text):

bcc2c68c abbbf1c3 f5b05d8e 7e73a4d2
7b7e1b20

K0 ⊕ opad:

5c5d5e5f 58595a5b 54555657 50515253
4c4d4e4f 48494a4b 44454647 40414243
7c7d7e7f 78797a7b 74757677 70717273
6c6d6e6f 68696a6b 64656667 60616263

(K0 ⊕ opa d) || Hash((Key ⊕ ipad)||text):

5c5d5e5f 58595a5b 54555657 50515253
4c4d4e4f 48494a4b 44454647 40414243
7c7d7e7f 78797a7b 74757677 70717273
6c6d6e6f 68696a6b 64656667 60616263
bcc2c68c abbbf1c3 f5b05d8e 7e73a4d2
7b7e1b20

HMAC(Key, Text) = Hash((K0 ⊕ opad) || Hash((Key ⊕ ipad)||text)):

4f4ca3d5 d68ba7cc 0a1208c9 c61e9c5d
a0403c0a

20-byte HMAC(Key, Text):

4f4ca3d5 d68ba7cc 0a1208c9 c61e9c5d
a0403c0a

A.2 SHA-1 with 20-Byte Key

Text: "Sample #2"

Key: 30313233 34353637 38393a3b 3c3d3e3f
40414243

K0: 30313233 34353637 38393a3b 3c3d3e3f
40414243 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000

K0 ⊕ ipad:

06070405 02030001 0e0f0c0d 0a0b0809
76777475 36363636 36363636 36363636
36363636 36363636 36363636 36363636
36363636 36363636 36363636 36363636

(Key ⊕ ipad)||text:

06070405 02030001 0e0f0c0d 0a0b0809
76777475 36363636 36363636 36363636
36363636 36363636 36363636 36363636
36363636 36363636 36363636 36363636
53616d70 6c652023 32800000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000248

Hash((Key ⊕ ipad)||text):

74766e5f 6913e8cb 6f7f108a 11298b15
010c353a

K0 ⊕ opad:

6c6d6e6f 68696a6b 64656667 60616263
1c1d1e1f 5c5c5c5c 5c5c5c5c 5c5c5c5c
5c5c5c5c 5c5c5c5c 5c5c5c5c 5c5c5c5c
5c5c5c5c 5c5c5c5c 5c5c5c5c 5c5c5c5c

(K0 ⊕ opad) || Hash((Key ⊕ ipad)||text):

6c6d6e6f 68696a6b 64656667 60616263
1c1d1e1f 5c5c5c5c 5c5c5c5c 5c5c5c5c
5c5c5c5c 5c5c5c5c 5c5c5c5c 5c5c5c5c
5c5c5c5c 5c5c5c5c 5c5c5c5c 5c5c5c5c
74766e5f 6913e8cb 6f7f108a 11298b15
010c353a

HMAC(Key, Text) = Hash((K0 ⊕ opad) || Hash((Key ⊕ ipad)||text)):

0922d340 5faa3d19 4f82a458 30737d5c
c6c75d24

20-byte HMAC(Key, Text):

0922d340 5faa3d19 4f82a458 30737d5c
c6c75d24

A.3 SHA-1 with 100-Byte Key

Text: "Sample #3"
Key: 50515253 54555657 58595a5b 5c5d5e5f
60616263 64656667 68696a6b 6c6d6e6f
70717273 74757677 78797a7b 7c7d7e7f
80818283 84858687 88898a8b 8c8d8e8f
90919293 94959697 98999a9b 9c9d9e9f
a0a1a2a3 a4a5a6a7 a8a9aaab acadaeaf
b0b1b2b3

Hash(Key):

a4aabe16 54e78da4 40d2a403 015636bf
4bb2f329

K0: a4aabe16 54e78da4 40d2a403 015636bf
4bb2f329 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000

K0 ⊕ ipad:

929c8820 62d1bb92 76e49235 37600089
7d84c51f 36363636 36363636 36363636
36363636 3636366 36363636 36363636
36363636 36363636 36363636 36363636

(Key ⊕ ipad)||text:

929c8820 62d1bb92 76e49235 37600089
7d84c51f 36363636 36363636 36363636
36363636 36363636 36363636 36363636
36363636 36363636 36363636 36363636
53616d70 6c652023 33

Hash((Key ⊕ ipad)||text):

d98315c4 2152bea0 d057de97 84427676
2a1a5576

K0 ⊕ opad:

f8f6e24a 08bbd1f8 1c8ef85f 5d0a6ae3
17eeaf75 5c5c5c5c 5c5c5c5c 5c5c5c5c
5c5c5c5c 5c5c5c5c 5c5c5c5c 5c5c5c5c
5c5c5c5c 5c5c5c5c 5c5c5c5c 5c5c5c5c

(K0 ⊕ opad) || Hash((Key ⊕ ipad)||text):

f8f6e24a 08bbd1f8 1c8ef85f 5d0a6ae3
17eeaf75 5c5c5c5c 5c5c5c5c 5c5c5c5c
5c5c5c5c 5c5c5c5c 5c5c5c5c 5c5c5c5c
5c5c5c5c 5c5c5c5c 5c5c5c5c 5c5c5c5c
d98315c4 2152bea0 d057de97 84427676
2a1a5576

HMAC(Key, Text) = Hash((K0 ⊕ opad) || Hash((Key ⊕ ipad)||text)):

bcf41eab 8bb2d802 f3d05caf 7cb092ec
f8d1a3aa

20-byte HMAC(Key, Text):

bcf41eab 8bb2d802 f3d05caf 7cb092ec
f8d1a3aa

A.4 SHA-1 with 49-Byte Key, Truncated to 12-Byte HMAC
Text: "Sample #4"

Key: 70717273 74757677 78797a7b 7c7d7e7f
80818283 84858687 88898a8b 8c8d8e8f
90919293 94959697 98999a9b 9c9d9e9f

K0: 70717273 74757677 78797a7b 7c7d7e7f
80818283 84858687 88898a8b 8c8d8e8f
90919293 94959697 98999a9b 9c9d9e9f
a0000000 00000000 00000000 00000000

K0 ⊕ ipad:

46474445 42434041 4e4f4c4d 4a4b4849
b6b7b4b5 b2b3b0b1 bebfbcbd babbb8b9
a6a7a4a5 a2a3a0a1 aeafacad aaaba8a9
96363636 36363636 36363636 36363636

Key ⊕ ipad)||text:

46474445 42434041 4e4f4c4d 4a4b4849
b6b7b4b5 b2b3b0b1 bebfbcbd babbb8b9
a6a7a4a5 a2a3a0a1 aeafacad aaaba8a9

96363636 36363636 36363636 36363636
53616d70 6c652023 34

Hash((Key ⊕ ipad)||text):

bf1e889d 876c34b7 bef3496e d998c8d1
16673a2e

K0 ⊕ opad:

2c2d2e2f 28292a2b 24252627 20212223
dcdddedf d8d9dadb d4d5d6d7 d0d1d2d3
cccdcecf c8c9cacb c4c5c6c7 c0c1c2c3
fc5c5c5c 5c5c5c5c 5c5c5c5c 5c5c5c5c

(K0 ⊕ opad) || Hash((Key ⊕ ipad)||text):

2c2d2e2f 28292a2b 24252627 20212223
dcdddedf d8d9dadb d4d5d6d7 d0d1d2d3
cccdcecf c8c9cacb c4c5c6c7 c0c1c2c3
fc5c5c5c 5c5c5c5c 5c5c5c5c 5c5c5c5c
bf1e889d 876c34b7 bef3496e d998c8d1
16673a2e

HMAC(Key, Text) = Hash((K0 ⊕ opad) || Hash((Key ⊕ ipad)||text)):

9ea886ef e268dbec ce420c75 24df32e0
751a2a26

12-byte HMAC(Key, Text):

9ea886ef e268dbec ce420c75

IV. Conclusion

At the end we have come to the conclusion that the use of CHMAC will increase the security and the authentication of the message during the transmission of the message from the sender to the receiver end. This we have proved by the use of the example. This proposed model also require less security by the transmission system during it transmit, as the message is coded by using the cryptographic algorithm making it least vulnerable to the different attacks and can also be used by the help of other cryptographic algorithm other than the DES.

References

[1] Atul khate, Cryptographic and Network Security, TMH, 2007.

[2] B. Preneel and P. van Oorschot, \On the security of two MAC algorithms," Advances in Cryptology { Eurocrypt 96 Proceedings, Lecture Notes in Computer Science} Vol. 22,U. Maurer ed., Springer-Verlag, 1996.

[3] H. Dobbertin, \Cryptanalysis of MD4," Fast Software Encryption Workshop, Lecture Notes in Computer Sciences, vol. 1039, Springer Verlag, 1996, pp. 53-69.

[4] H. Dobbertin, \ The Status of MD5 After a Recent Attack", RSA Labs' CryptoBytes, Vol. 2 No. 2, Summer 1996.

[5] I. Damgºard, \A design principle for hash functions," Advances in Cryptology {Crypto 89 Proceedings, Lecture Notes in Computer Science Vol. 435, G. Brassared., Springer-Verlag, 1989.

[6] Mihir Bellare¤ Ran Canettiy Hugo Krawczykz Keying Hash Functions for Message Authentication Paper published in 1996 Advances in Cryptology { Crypto 96 Proceedings,Lecture Notes in Computer Science Vol. 1109, N. Koblitz ed., Springer- Verlag, 1996.

[7] M. Bellare, R. Guferin and P. Rogaway, \XOR MACs: New methods for message authentication using ¯nite pseudorandom functions," Advances in Cryptology {Crypto 95 Proceedings, Lecture Notes in Computer Science Vol. 963, D. Coppersmithed., Springer-Verlag, 1995.

[8] M. Bellare, R. Canetti and H. Krawczyk, \Pseudorandom functions revisted: the cascade construction and its concret security," Proceedings of the 37th Symposium on Foundations of Computer Science, IEEE, 1996.

[9] National Institute for Standards and Technology, \Digital Signature Standard (DSS)", Federal Register, Vol. 56, No. 169, August, 1991.

[10] O. Goldreich, S. Goldwasser and S. Micali, \How to construct random functions,"Journal of the ACM, Vol. 33, No. 4, 210{217, (1986).

[11] R. Atkinson, \Security Architecture for the Internet Protocol", IETF Network Working Group, RFC 1825, August 1995.

[12] R. Atkinson, \IP Authentication Header", IETF Network Working Group, RFC 1826, August 1995.

[13] William Stalling, Cryptography and Network Security, PHI, 2004.