A Message Digest System Using Key Concept and Modulus Operations

Suman Deb
National Institute of Technology
Agartala, India

Gautam Rakshit
National Institute of Technology
Agartala, India

Pritam Rakshit
National Institute of Technology
Agartala, Indi

ABSTRACT
Message Digest is the promising methodology of data integrity in the concerned areas of computer communication. Different techniques of message digest are available and used for checking the data integrity and indirectly used in digital signatures for authentication of the transmitted information. Several research works are going on to make the computational process of message digest generation more complex to the unauthorized users. This paper introduces the procedure for generating message digest (Named as SGP-MD) using key in its algorithms so that the digest itself can directly act as a Message Authentication Code (MAC) and hence removing the need of separate encryption/decryption algorithm to create the MAC from the digest. The implementation of the proposed algorithm suffices for both data integrity and authentication purpose

Keywords
Data Authentication and Integrity, Data security, key stream, Message Digest.

1. INTRODUCTION
Message digest is a fingerprint or the summary of the original message whose size is much shorter than that of the original message [1]. 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 reach the receiver*). Developing a message digest system must meet with three basic requirements as

i) Given a message it should be very easy to find its corresponding message digest. Also for a given message the message digest should always be the same.

ii) Given a message digest it should be very difficult to find the original message for which the digest was created.

iii) Given any two messages if anybody calculate their message digests, the two message digest must be different.

If any two messages produce the same message digest then it violates the basic principle of creating message digest, called '*a collision'* .That is if two message digest collide [5] they meet at the same digest.

Message digest algorithms usually produce a digest of length ranging from 128 bits to 256 bits. Mathematically these algorithms has a Hash function [10] that can map bit strings of arbitrary finite length into strings of fixed length. A hash value is generated by a function

$$h = H(M)$$

Where M is a variable-length message and H (M) is is the fixed-length hash value. The purpose of a hash function is to produce a fingerprint of a file, message or other block of data. To be useful for message authentication, a hash function must satisfy the following properties:

1- H can be applied to a block of data of any size.

2- H produces a fixed-length output.

3- H(x) is relatively easy to compute for any given input x.

4- One-way: for any given code h, it is computationally infeasible to find x such that H(x) = h.

5- Strong collision resistance: it is computationally infeasible to find any pair (x, y) such that H(y) = H(x).

The strength of a hash function against brute-force attacks [l] depends only solely on the length of hash code produced by the algorithm. Table I Summarizes the level of effort required producing a birthday or square root attack (referred as strength of hash code) for different types of hash functions, assuming m-bit result.

Table I. Comparison of hash-code strength between different types of hash function

Type of hash function	Strength of hash function
One-way	$2^{\wedge m}$
Weak collision resistance	$2^{\wedge m}$
Strong collision resistance	$2^{\wedge (m/2)}$

Following figure shows the general structure of a typical secure hash function.

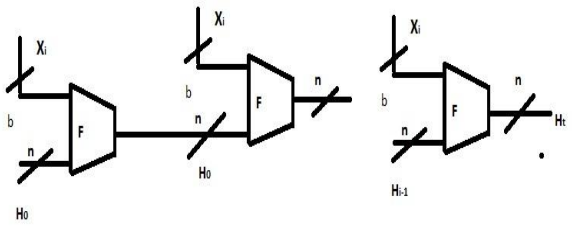


Fig 1.The general structure of a typical secure hash function

Where H=chaining variable, $X_i = i^{th}$ input block, F=compression function, i =number of input blocks, n=length of hash code, b=length of input block.

2. RELATED STUDIES

The study of some of the well known algorithms as noted in [3][4][6][7][8][9] such as MD4, MD5, SHA-1, SHA-256 etc. reveals that the strength of the algorithms lies in the fact that the algorithms should *provide* as much of complexity and randomness as possible to the algorithm, so that no two message digests produced by the algorithm on any two message are equal. In general all the message digest algorithms have a property that every bit of the message digest is some function of every bit in the input. Almost all hash functions are iterative processes, which hash inputs of arbitrary length by processing successive fixed-size blocks of the input. The input X is padded to a multiple of the block length and subsequently divided into t blocks X_1 through X_t. The overall structure of a typical secure hash function is indicated in Fig.1. The hash function h can then be described as follows:

H_0 = initial n-bit value

H (i) = F (H_{i-1}, Xi), 1<i<t, H(X) = H (t).

The first constructions for hash functions were based on block ciphers (such as DES).Although some trust has been built up in the security of these proposals, their software performance is not very good, since they are typically from 2 to 4 times slower than the corresponding block cipher. The most popular hash functions, which are currently used in a wide variety of applications, are the custom designed hash functions from the family MD4 which was proposed by R.Rivest [3].

It is a very fast hash function tuned towards 32-bit processors. Because of unexpected Vulnerabilities identified in [11][12] (namely collisions for two rounds out of three), a strengthened version of MD4 was designed which is called MD5 [4]. MD5 is slightly slower than MD4, but it is more conservative in design. It was being implemented fast into products. MD5 is probably the most widely used hash function, in spite of the fact that the compression function of MD5 is not collision resistant [10].

3. PROPOSED MESSAGE DIGEST ALGORITHM

In this paper, a new idea of incorporating key in generating a message digest is provided so that the message digest itself act as MESSAGE AUTHENCATION CODE (MAC). Here two integers are considered as the first level key K1 that is to be communicated between the sender and the receiver through some secret channel. From this K1 the second level key is produced which is a list of 32 byte. These 32 byte list is created and stored in an intermediate array which will be used as a storage of intermediate results throughout the algorithm and also stores the final result (*message digest*). The algorithm has fixed the maximum input message size less than 2^64 bits length[3][4]. The output is a message digest of 256 bits in length.

3.1 Procedure to generate the second level key

[The algorithms implemented by considering byte as a positive number from 0 to 255 *(as c# language specification)]*

Input: i) Key k_1, a set / array of two unsigned integers.

K_1 = {65, 267}

ii) Intermediate Byte array namely Inter [] of size 32.

Output: Filling the intermediate array Inter [32] with 32 different byte values using key1 **.**These bytes will be used for triggering the message digest algorithm.

Steps:

Step 1. Let Inter [0] = (key1 [0] %256.

Step 2 Let Integer variable K=key1 [1]

Step 3.Let Integer Variable R=0

Step 4.Repeat through step 8 for I=1 to 31

Step 5.R=K+ Inter [I-1]

Step 6.If (R>256)

Step 7.R=R%256

Step 8.Inter[i] = Convert To Byte(R)

Step 9.Repeat through step15 for I=0 to 7

Step10.Byte Variable P= Convert to Byte ((Inter[i*4] AND Inter[i*4+1]) XOR ((NOT(Inter[i*4+2])) AND Inter[i*4+3]))

Step 11. Repeat through step 15 for J=0 to 3

Step 12.R=Inter[i*4+j] + P

Step 13.If (R>256)

Step 14.R=R%256

Step15.Inter [i*4+j]=Convert To Byte(R)

Step 16. Stop

3.2 Procedure to generate the message digests

Input: a.) Intermediate **Inter [32]** initialized with second level key using procedure of 3.1.

b) An input message maximum size less than2^64

c)A Temporary integer array of size 32. let its name be Temp [32]

d) A process P of type Byte. This will be

Output: Message digest of length 256 bits.

Steps

Step1.The original message is padded such that the length of the message bits is 64 bits less than a multiple of 256 bits. Then append length of the original message excluding the padding at end of the padded message as 64 bit block. The padding consists of as many bytes as required and this bytes are supplied repeatedly from the first time initialized **Inter [32]** list. Note that padding is always added even if the message is already 64 bits less than multiple of 256.

Step 2.Read 32 bytes (256 bits) at time and store it in temp array for each 32 bytes Repeat through step x.

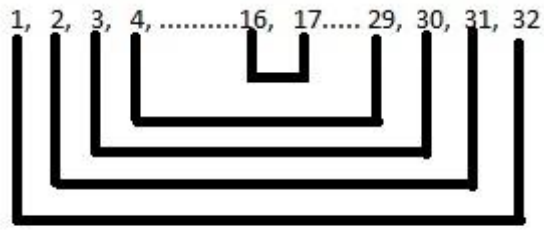
Step 3.Repeat through step 3.3 for I =0 to 31

Step3.1.Temp [I] =Temp [I] + Inter [I];

Step 3.2. If Temp [I]>256 then.

Step 3.3.Temp [I]=Temp[I]%256.

Step 4.Perform a special swap operation on the Temp array to add randomization. Following figure shows the pattern.



SWAP ORDER

Step 4.1.Let Variable F=0, Variable R=31

Step 4.2.Repeat through step 4.6 for I = 0 TO 15

Step 4.3.T=Temp [F].

Step 4.4.Temp [F] =Temp[R].

Step 4.5.Temp[R] =T.

Step 4.6.F=F+1 and R=R-1.

Step 5

Consider the Temp array and the Intermediate array Inter [] into a group/block of 8 each of size 4 bytes.

Step 6

Repeat through step 11 for each group I=0 to 7.

Step 7

Repeat through step 11 for each round J=0 to 3.

Step 8

If (J=0) then

*P = (Temp [I*4+0] AND Temp [I*4+1])OR*

*(NOT (Temp [I*4+2]) AND (Temp [I*4+3]))*

If (J=1) then

*P= (Temp [I*4+1] XOR Temp [I*4+2] XOR Temp [I*4+3]) OR Inter [I*4+0].*

If J=2 then

*P=(Temp[I*4+1] AND Temp[I*4+2])OR(Temp[I*4+1] AND Inter[I*4+3])OR(Temp[I*4+2]AND Temp [I*4+3])).*

If J=3 then

*P=Inter [I*4+1] XOR Temp [I*4+2] XOR Temp [I*4+3])).*

Step 9

Repeat through step 11 for K=0 to 3

Step 10

*Temp [I*4+K] =Temp [I*4+K] +P*

Step 11

If Temp [I*4+K]>256 then

Temp [I*4+K] =Temp [I*4+K] %256

Step 12 Repeat through step 13 for I =0 to 31

Step 13 Inter [I] =Temp [I]

Step 14 Stop

4. KEY STRENGTH ANALYSIS

4.1 Analysis for Brute-force attack

1. First level key (K_1) comprises of two unsigned integers each occupying 4 bytes. Therefore K_1 have total bit length of 64 (32+32) bits. Here key2 is not considered in strength analysis as the second level key generation algorithm is open. But even then the hacker has to find the correct key2.Total numbers of possible keys having key size of 64 bits are as follows:

2^64 = 18.446744073709551616 x 10^18 possible keys.

Now assume that a hacker have a very fast computer using which he/she can execute the decryption algorithm in 1 micro second for all possible key trials. Even if he tries half the set of keys then also he is quite successful in decrypting.

But then also the hackers require more than one year decrypting the cipher text which is shown as below:

In one second = 1000000 possible key trials

In one hour = 36 x 10^8 possible key trials

In one day = 864 x 10^8 possible key trials

In one year = 3.1536 x 10 ^13 possible key trials (less than half of the total key set).

The Intermediate array Inter [] is analogous to the chaining variables used in some of the well known algorithms such as MD5, SHA-1, SHA-256 etc. But all these algorithms initialize the chaining variables with predefined constants to start the algorithm. But in this proposed paper the intermediate array is never initialized with predefined constants. The initialization takes place in a random basis using level one key key1[X,Y].Following table shows some of the outputs of algorithm of III(A) on some closely related keys of level one key key1[X,Y]. It can be observer from the table that for a slight difference in any two keys the algorithm III(A) produces a vastly different and random **Inter[]** list. Compare first three rows of the table as one class of closely related keys and the last two as another class of closely related keys. It is clear that even a single bit change in two values(X,Y) of K1 the respective derived Inter[] list differs with a great extent. The Inter [] list is the implicit second level key derived from level one key1[X, Y].

Table 1 Test Result of algorithm III (A)

Key1[X,Y]	**Intermediate Inter[] first time values for different values of key1(some Test results)**
K1[65,267]	161 ,172 ,183 ,98 ,209 ,220 ,231 ,142 ,41 ,52 ,63 ,186 ,169 ,180 ,191 ,230 ,209 ,220 ,231

	,18 ,33 ,44 ,55 ,62 ,169 ,180 ,191 ,106 ,137 ,148 ,159 ,150 .
K1[61,267]	65 ,76 ,87 ,98 ,73 ,84 ,95 ,106 ,41 ,52 ,63 ,74 ,161 ,172 ,183 ,194 ,209 ,220 ,231 ,242 ,41 ,52 ,63 ,74 ,169 ,180 ,191 ,202 ,209 ,220 ,231 ,242 .
K1[62,267]	65 ,76 ,87 ,98 ,213 ,224 ,235 ,246 ,41 ,52 ,63 ,74 ,165 ,176 ,187 ,198 ,209 ,220 ,231 ,242 ,37 ,48 ,59 ,70 ,169 ,180 ,191 ,202 ,213 ,224 ,235 ,246 .
K1[61,62]	188 ,250 ,56 ,118 ,180 ,242 ,48 ,110 ,156 ,218 ,24 ,86 ,164 ,226 ,32 ,94 ,124 ,186 ,248 ,54 ,116 ,178 ,240 ,46 ,92 ,154 ,216 ,22 ,68 ,130 ,192 ,254 .
K1[61,60]	182 ,242 ,46 ,106 ,150 ,210 ,14 ,74 ,118 ,178 ,238 ,42 ,86 ,146 ,206 ,10 ,182 ,242 ,46 ,106 ,150 ,210 ,14 ,74 ,118 ,178 ,238 ,42 ,86 ,146 ,206 ,10 ,
K1[61,63]	185 ,248 ,55 ,118 ,177 ,240 ,47 ,110 ,169 ,232 ,39 ,102 ,161 ,224 ,31 ,94 ,153 ,216 ,23 ,86 ,145 ,208 ,15 ,78 ,137 ,200 ,7 ,70 ,129 ,192 ,255 ,62 .

5. ALGORITHM STRENGTH ANALYSIS

The proposed Message digest fulfills the following requirements and hence justifies its strength:

1.The message digest is irreversible i.e. from a given digest it is hard to derive the original message, mathematically it is not possible because the process P operations are irreversible and more over modulus operation make the message digest harder to be reversible.

2. The algorithm also has sufficient complexity and randomness like other standard algorithms. Swapping is introduced before the process **P** operations. Here in this message digest system initialization of intermediate list Inter [] (analogous to chaining variables as in [3][4][6][7]) takes place in a random basis using level one key k1[X, Y] while it is initialized by some predefined constants in MD4, MD5, SHA-1, SHA-256 .This adds authentication to the message digest and *hence by passing the requirement of separate encryption/decryption algorithm for creating MAC(Message authentication code).Here the message digest itself acts as MAC.*

3. The heart of the message digest algorithm starts from step 8.From step 8 onwards 128 iterations are performed on the total block of 256 bits. .

4. The possibility that two messages produce the same message digest is in the order of 2^64 operations.

5. Given a message digest to find the original message can lead up to 2^256 operations.

Some of the test results are shown in the following table. To show the testing here Key1[X, Y] =K1 [61][63] is considered here for all the test cases. It is here considered that the input messages are already padded message.

6. The bitwise circular shift operation as noted in [3][4][6][7] has been removed here for faster execution because circular shift consumes good number of swapping operations.

Table 2: Test results of the message digest algorithm.

Message(1)	**126**, 186, 250, 57, 120, 183, 246, 53, 116,179, 242, 49, 112, 175, 238, 45, 108, 171, 234, 41, 104, 167, 230, 37, 100, 163,226, 33,96,159,222,32,171,234,41,104, 167, 230, 37, 100, 163, 226,242, 49, 112, 175, 238, 45, 108, 171, 234, 41, 104, 167, 230, 37, 100, 163, 226, 33, 96, 159, 222,32
Digest1	234 ,35 ,99 ,162 ,225 ,32 ,95 ,158 ,25 ,88 ,151 ,214 ,177,240 ,47 ,110 ,201,8 ,71 ,134 ,116 ,179 ,33 ,96 ,233 ,40 ,103 ,166 ,112 ,175 ,238 ,48 .
Message(2)	**127,** 186, 250, 57, 120, 183, 246, 53, 116,179, 242, 49, 112, 175, 238, 45, 108, 171, 234, 41, 104, 167, 230, 37, 100, 163,226, 33,96,159,222,32,171,234,41,104, 167, 230, 37, 100, 163, 226,242, 49, 112, 175, 238, 45, 108, 171, 234, 41, 104, 167, 230, 37, 100, 163, 226, 33, 96, 159, 222,32
Digest2	**243 ,43 ,107 ,170** ,225 ,32 ,95 ,158 ,25 ,88 ,151 ,214 ,177 ,240 ,47 ,110,201,8 ,71 ,134 ,116 ,179 ,33 ,96 ,233 ,40 ,103 ,166,**179,242 ,49 ,115** .
Message(3)	**127**, 186, 250, **60**, 133, 183, 246, 53, 116,179, 242, 49, 112, 175, 238, 45, 108, 171, 234, 41, 104, 167, 230, 37, 100, 163,226, 33,96,159,222,32,171,234,41,104, 167, 230, 37, 100, 163, 226,242, 49, 112, 175, 238, 45, 108, 171, 234, 41, 104, 167, 230, 37, 100, 163, 226, 33, 96, 159, 222,32
Digest3	**243 ,43 ,107 ,170 ,62 ,112 ,175 ,238** ,25 ,88 ,151 ,214 ,177 ,240 ,47 ,110 ,201 ,8 ,71 ,134 ,116 ,179

	,33 ,96 **,77 ,140 ,203 ,10 ,179 ,242 ,49 ,115 .**
Message(4)	**127, 187**, 250, 60, 255, 183, 246, 53, 116,179, 242, 49, 112, 175, 238, 45, 108, 171, 234, 41, 104, 167, 230,37,100,163,226,33,96,159,222,32,171,234,41, 104, 167, 230, 37, 100, 163, 226,242, 49, 112, 175, 238, 45, 108, 171, 234, 41, 104, 167, 230, 37, 100, 163, 226, 33, 96, 159, 222,32
Digest4	**124 ,180 ,244,54,56 ,240,47 ,110** ,25 ,88 ,151 ,214 ,177 ,240 ,47 ,110 ,201 ,8 ,71 ,134 ,116 ,179 ,33 ,96 **,89 ,152 ,215 ,22 ,79 ,142 ,205 ,15**

The message(1) and the digest1 can be set as a reference point. The other three messages and their corresponding digests are generated after making slight changes in the reference message(1) to observe how much the digest differs from the reference digest...The message(2) differs from reference message(1) by a single bit (126 i.e. (01111110) and 127 i.e. (01111111) differs by a single bit) though the digest2 differs by 8 bytes (more than one bits) from digest1.This strengthen the fact that even a small change in the message results in drastic change in its message digest.

Similarly it can be observed in message(3) that for a change of 3-bits a total of 16 bytes differ from the reference message digest digest1.Another strength of the algorithm is that the change in the output message digest from a reference point is not related to change in the number of bit in the message from a reference message. This fact can again be proved from the message(4) and its digest4.It can be observed that for a chance of 8-bits in the message(4) with respect to the reference message(1) the message digest4 differs by 16 bytes same as the case of message(3).

Following is a comparison of the proposed algorithm with different versions of SHA.

Table 3 Comparison of SGP-MD vs. SHA

Parameter	SHA-1	SHA-256	SHA-384	SHA-512	SGP-MD
Message Digest size(in bits)	160	256	384	512	256
Message Size(in bits)	<2^64	<2^64	<2^128	<2^128	<2^64
Block size (in bits)	512	512	1024	1024	256
Word size (in bits)	32	32	64	64	32
Iterations	80	64	80	80	128

From the above table it is inferred that SGP-MD imposes more iterations on lesser block size and word size. Therefore it adds more complexity and randomness to the algorithm.

6. CONCLUSION

In general message digest purely solves the integrity issue of security. In order to add authentication with it one have to encrypt the message digest before transmitting it .This technique is known as MAC(Massage Authentication Code).To achieve this MAC apart from message digest algorithm, which itself is a complex algorithm one extra algorithm needs to be used i.e. encryption /decryption. To reduce this extra over head it is proposed here to create a message digest using a key keeping message digest properties intact and this will ultimately save lot of computational overheads. Hence the proposed message digest system can be very useful in modern integrity sustaining practices as well as for authentication .The level one key key1 is used to generate the random chaining variables which are used in the creation of message digest. The algorithm performs 128 operations on a block of 256 bits. Which is quite complex and adds randomness. The possibility that two messages produce the same message digest is in the order of 2^64 operations. Given a message digest to find the original message can lead up to 2^256 operations.

7. REFERENCES

[1]. Atul Kahate: "Cryptography and Network security"

Tata McGraw Hill Education Pvt. Ltd (2nd edition 2003).

[2].Stallings, W.: 'Cryptography and Network Security: Principles and Practice' (Prentice Hall Inc., 2nd edn.1999).

[3]. Rivest,R. L.: 'The MD4 message-digest algorithm'. Proc. Crypto'90,LNCS 537, 1991, pp. 303-311 [4]. NIST, "Secure Hash Standard," FIPS PUB 180, May. 1993.

[4]. R.L. Rivest.: 'The MD5 Message Digest Algorithm'. RFC 1321, 1992.

[5]. X. Wang, H. Yu and Y. L. Yin "Efficient Collision Search Attacks on SHA-0" eprint 2005, http://eprint.iacr.org/2005/391.pdf

[6]. Federal Information Processing Standard (FIPS): 'Publication 180-1, Secure Hash Standard (SHS)', U.S. Doc/NIST, April 1995.

[7]. Federal Information Processing Standard (FIPS): 'Publication 180-2, Secure Hash Standard (SHS)', U.S. Doc/NIST, May 2001.

[8]. Integrity Primitives for Secure Information Systems: Final Rep. of RACE Integrity Primitives Eval. RIPE-RACE 1040, LNCS 1007, 1995Rump Session, Aug. 2004.

[9]. Zheng,Y., Pieprzyk, J. and Sebery,J.: 'HAVAL -A One-Way Hashing Algorithm with Variable Length of Output'. Proc. Auscrypt'92, LNCS 718, 1993, pp. 83-104.

[10]. Abdul Hamid M.Ragab, Nabil A. Ismail, Senior Member IEEE, and Osama S. FaragAllah "An Efficient Message Digest Algorithm for Data Security".

[11]B. den Boer, A. Bosselaers, "An Attack on the Last Two Rounds of MD4," Advances in Cryptology, proc. Crypto'91, LNCS 876, J. Feigenbaum, Ed, Spriager-Verlag, 1992, pp. 194-203.

[12] H. Dobbertin,"Cryptanalysis of MD4,"Fast Software Encryption, LNCS 1039, D. Gollmann, Ed., Springer-Verlag, 1996.