Message Integrity in the World Wide Web: Use of Nested Hash Function and a Fast Stream Cipher

Sheena Mathew
Department of Computer Science
Cochin University of Science and Technology
Kochi, Kerala, India
sheenamathew@cusat.ac.in

K. Paulose Jacob
Department of Computer Science
Cochin University of Science and Technology
Kochi, Kerala, India
kpj@cusat.ac.in

***Abstract*-The focus of this work is to provide authentication and confidentiality of messages in a swift and cost effective manner to suit the fast growing Internet applications. A nested hash function with lower computational and storage demands is designed with a view to providing authentication as also to encrypt the message as well as the hash code using a fast stream cipher MAJE4 with a variable key size of 128-bit or 256-bit for achieving confidentiality. Both nested Hash function and MAJE4 stream cipher algorithm use primitive computational operators commonly found in microprocessors; this makes the method simple and fast to implement both in hardware and software. Since the memory requirement is less, it can be used for handheld devices for security purposes.**

I INTRODUCTION

Due to the prospering use of Internet applications like e-commerce, ensuring confidentiality, integrity and authenticity of information have acquired increased importance [1]. When two parties are communicating over an insecure channel, they need a method by which the information sent by the sender can be accepted as confidential, unmodified and authentic by the receiver. The confidentiality of the message can be achieved by encrypting the message by the symmetric key algorithms in cryptography, which are faster and efficient and use the same key for both encryption and decryption of data. The integrity of the message can be verified by hash functions. Hash function is a function of all the bits of the message. It accepts a variable size message as input and produces a fixed size output as the hash code. A change in any bit or bits in the message results a change in the hash code [2] thus providing an indication of message tampering. When A sends a message to B, it appends to the message the hash code, which is computed using the hash function. After receiving the message B re-computes the hash code using the same hash function and compares with the original hash code. If both are same then B can assure that the message has started off from the intended sender and it has not been tampered with, during the transmission.

Common uses of Hash functions include authorization of financial transactions, mobile communications (GSM and GSPP) and authentication of Internet communications with SSL/TLS and IPSec. It is also used as a pseudo random function. Hash function provides a deterministic mechanism for generating random seeming bit streams from some input source without disclosing any information about the input. It can also be used to give protection against viruses. Viruses typically modify the host files that they infect, and hence one way of virus detection involves checking files for signs of unauthorized modification by computing the authentication tags from each file [3].

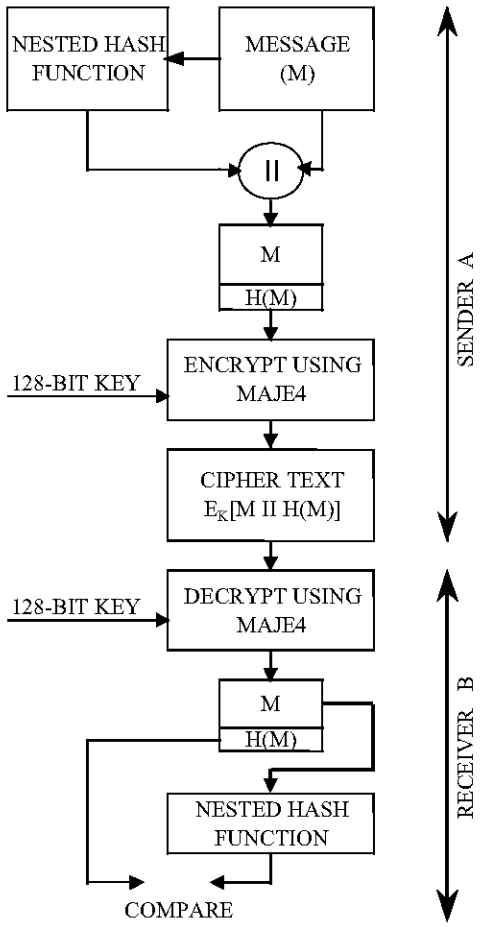

Fig. 1. Use of combined hash code and encryption.

As shown in Fig. 1, sender A uses the nested hash function to compute the hash code H(M) of the message M and appends it to the message M. Using the 128-bit key K and the

fast stream cipher algorithm MAJE4 the message and the hash code are encrypted as $E_k[M \| H(M)]$ and sent to the receiver B. Using the same key K and the fast stream cipher algorithm MAJE4 the cipher text is decrypted back to produce the message and the hash code. Now B re-computes the hash code of the received message using the same nested hash function. Thus B validates the integrity of the message by comparing the hash code received from A with that generated by B. If both hash codes are same then the transmission has happened securely.

II DESIGN OBJECTIVES

The following factors are considered for the design [4] of hash function and the stream cipher MAJE4.

A. Hash Function

1) Hash functions can be applied to messages of any length.

2) It produces an output of fixed length.

3) For any given x, it is easy to compute H(x) making both hardware and software implementation easy.

4) For any given value h, it is computationally infeasible to find x such that H(x) = h.

5) For any given block x, it is computationally infeasible to find y≠x with H(y)=H(x).

6) It is computationally infeasible to find any pair (x,y) such that H(x)=H(y).

B. MAJE4

1) The encryption sequence can have a large period.

2) The key stream can approximate the properties of a true random stream.

3) MAJE4 is suitable for hardware or software implementation. It uses only primitive computational operations commonly found in microprocessors.

4) It is simple and fast. It uses simple algorithm, which is easy to implement and eases the task of determining the strength of the algorithm.

5) Low memory requirements make it suitable for handheld type devices with restricted memory.

6) Mixed operators are used for the design of MAJE4. The use of more than one arithmetic and / or Boolean operator complicates cryptanalysis. Primitive operators like + and ^ are used since these operators do not commute and hence cryptanalysis becomes again more difficult.

7) Variable number of rounds are used. An increase in the number of rounds increases cryptanalytic strength.

III NESTED HASH FUNCTIONS

The Merkle-Damgaard model is a good one for the design of hash functions[5,6]. This model simplifies the management of large inputs and the production of a fixed length output using a function F. The message is viewed as a collection of m bit blocks. M= M[1].....M[n] with M[i]=m bits for i=1,2....n. Assume the length |M| of M as a multiple of m bits, which can be achieved by a suitable padding. Enough number of zeros are added to bring the length of message to multiple of m bits. The blocks are then processed sequentially using the function F. The result of the hash function till then and the current message block are taken as the inputs. This operation is repeated over the entire message blocks to find the hash code of the message M at the end.

The following steps are used to compute the hash code.

1) The message is viewed as a collection of 64-bit blocks. M= M[1],M[2],.....M[n] with M[i]=64 bit for i=1,2,.....n.

2) Check whether the length |M| of M is a multiple of 64 bits and whether n is an even number, if not suitably append enough zeros to bring the length to a multiple of 64 bits and to make n even.

3) Apply the first function F1 which is the add operation to the consecutive blocks. (MB[1] =M[1] + M[2], MB[2] =M[3] + M[4] and so on till MB[n/2] = MB[n-1]+MB[n].)

4) Apply the second function F2 which is an XOR operation, to the random initial value and to MB[1] and form the initial hash code. Then F2 is applied again to the initial hash code and to MB[2] to form the next hash code and so on . Finally apply F2 on the result of the hash code obtained so far and to MB[n/2] to form the final hash code H(M) of 64 bit length.

5) Now H(M) is added with M as the authentication tag.

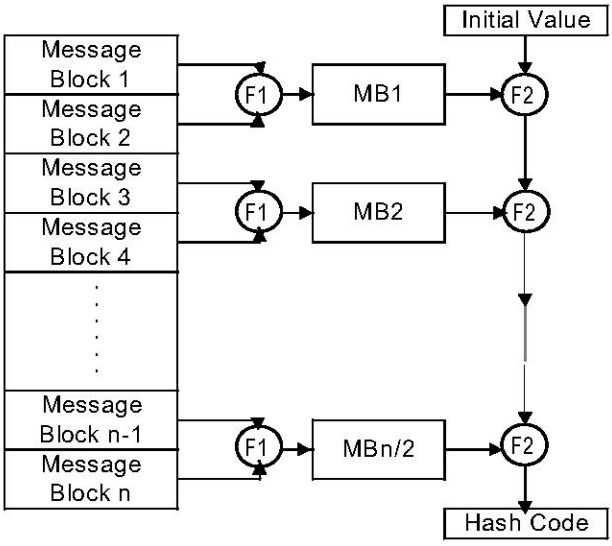

Fig. 2. Model of a Nested Hash Function

Fig. 2 represents the steps explained above. The random initial value used in step 4 provides message integrity protection and authentication to the hashing process to compute the hash of the initial message. The recipient can verify that the message is authentic by using the same random initial value, which was used to compute the hash code of the

message. If these hashes match, then the message is believed to have arrived unchanged from the sender. Thus the initial random value prevents attackers from making undetectable changes to the message. As specified in the design factors of hash function, message of any length can be considered as the input while the output hash code is of fixed 64-bit length. The initial value used as K in (1) is random and hence the attackers will not be able to predict the initial value easily. The functions F1 and F2 in (1) are ADD and XOR operations [7] which are easy to implement both in hardware and software. At the same time the nested usage of operators + and ^ complicates cryptanalysis. Mainly the security of the message authentication mechanism depends on the cryptographic properties of the hash function H. Here the non-linearity is obtained when functions F1 and F2 are nested. This provides added security.

$$\text{That is } H(M) = F2K(F1(M)) \qquad (1)$$

It is also observed that the length in bits of a message authentication code is directly related to the number of trials that an attacker has to perform before a message is accepted. For a message authentication value of bit length m, the attacker has to perform on average 2^{m-1} random online message authentication code verifications. The minimum reasonable length for the message authentication code is 32 bits; this corresponds to about 2 billion trials. Here more appropriate 64 bits blocks are considered.

IV A FAST STREAM CIPHER - MAJE4

The following steps are used in MAJE4 stream cipher [8].
One can choose either a 128-bit key or a 256-bit key.
128-bit key: The first four 32 bit words, ie. $key_{[0]}$, $key_{[1]}$, $key_{[2]}$ and $key_{[3]}$ are considered for storing the key.
256-bit key: The key is stored in eight 32 bit words $key_{[0]}$, $key_{[1]}$, $key_{[2]}$, $key_{[3]}$, $key_{[4]}$, $key_{[5]}$, $key_{[6]}$ and $key_{[7]}$.

Steps:

1) Assign the key length k1 either as 128-bit or 256-bit.
2) if kl = 128 then div=4
else div=8
3) if kl = 128 then consider two lsb's of $key_{[0]}$, find the decimal equivalent of these two lsb's and store in the variable 'in'.
else
4) if kl = 256 then consider three lsb's of $Key_{[0]}$, find the decimal equivalent of these three lsb's and store it in a variable 'in'.
5) ran = $key_{[0]}$ ^ $key_{[in]}$
6) if kl = 128 then consider two lsb's of ran, find the decimal equivalent of that and store in the variable 'in1'.
7) if kl = 256 then consider three lsb's of ran, find the decimal equivalent of that and store in the variable 'in1'.
8) check the 16^{th} bit in ran,
9) if it is 1 then
newran=($key_{[in1]}$ + $key_{[in1+1 \bmod div]}$) ^ ($key_{[in1+2 \bmod div]}$ + $key_{[in1+3 \bmod div]}$)
else
newran=($key_{[in1]}$ ^ $key_{[in1+1 \bmod div]}$) + ($key_{[in1+2 \bmod div]}$ ^ $key_{[in1+3 \bmod div]}$)
10) The output 32-bit word is newran, which can be used to XOR with the corresponding word in the plain text
11) Advance all the keys as
$key_{[i]} = key_{[i]} * key_{[i]} + key_{[i]} >> 20$
12) go to step 3.

This MAJE4 is a 128-bit or 256-bit key algorithm and the randomness property of the stream cipher is analyzed by using the five statistical tests like frequency test, serial test, poker test, runs test and autocorrelation test [9]. All the five statistical tests are passed by this generator for all the random streams produced. Hence MAJE4 algorithm can be used very well for encrypting the message and the hash code as shown in Fig.1.

V USE OF HASHCODE AND MAJE4

The following are the steps performed for obtaining the confidentiality and authentication using MAJE4 and nested hash function.

1) Sender encrypts the message M as well as the hash code H(M) using 128-bit key K and the fast stream cipher algorithm MAJE4, then sends it to the receiver.
2) Receiver decrypts the message as well as the hash code using the same 128-bit key K and MAJE4 algorithm.
3) Receiver re-computes the hash code H(M) over the message M and checks whether it matches with the received hash code.
4) If it matches with the hash code, then the message can be considered to have reached securely. Otherwise it can be understood that some distortion has happened.

VI RESULTS

TABLE I
TIME TAKEN FOR ENCRYPTION OR DECRYPTION OF FILES OF VARIOUS SIZES USING MAJE4

File size of plain text (bytes)	File size of cipher text (bytes)	Time taken (Sec.)	
		Encryption	Decryption
135094	135094	0.05	0.05
270728	270728	0.10	0.10
541608	541608	0.20	0.20
812440	812440	0.30	0.30
1082953	1082953	0.40	0.40

Tables I to III show the results of time requirements for the MAJE4 and nested hash code when run with plain texts of different sizes. The memory sizes of the plain text to be encrypted, the cipher text, the time taken for encryption and decryption and the time taken for producing the hash code are given. The evaluation is done using Pentium IV processor, Linux operating system and C compiler.

TABLE II
TIME TAKEN FOR PRODUCING THE HASH CODE OF FILES OF VARIOUS SIZES USING NESTED HASH FUNCTION

File size of plain text (bytes)	Time taken for producing the hash code (Sec.)
135094	0.01
270728	0.02
541608	0.04
812440	0.06
1082953	0.08

TABLE III
TOTAL TIME TAKEN FOR PRODUCING THE HASH CODE AND ENCRYPTION/DECRYPTION OF FILES OF VARIOUS SIZES USING NESTED HASH FUNCTION AND MAJE4.

File size of plain text (bytes)	Total time taken for producing the hash code & recomputing the hash code. (Sec.)	Total time taken for encryption and decryption.(Sec.)
135094	0.02	0.10
270728	0.04	0.20
541608	0.08	0.40
812440	0.12	0.60
1082953	0.16	0.80

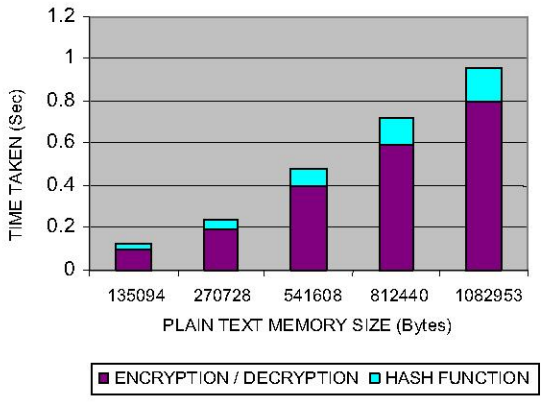

Fig 3. Total time taken for message authentication and encryption / decryption

In table III it can be seen that only $1/5^{th}$ of additional time is required for producing the hash code along with encryption and decryption. More over if the message size is reasonably small or upto about 135 kilobytes then the time taken for producing the hash code is negligible. For large messages the additional time requirement is very less as shown in Fig. 3. The memory size required for executable code for nested hash code is 5899 bytes and for MAJE4 it is 6254 bytes. Hence a total of nearly 12 Kilobytes memory is enough for providing both authentication and confidentiality of messages.

VII CONCLUSION

From the analysis of results it is concluded that message authentication can be achieved with confidentiality of message by using a very small increase in time even for too large messages. The time required for messages with a memory of upto 135 Kbytes is found negligible. Also the additional memory size needed for implementing the hash function is only 5899 bytes. Because of the low memory requirement, this hash code can be very well used in handheld devices like mobile phones, personal digital assistants (PDA), etc. for authentication purposes. Since it is faster, it can be used for applications that require message integrity and encryption / decryption of stream of data sent through the Internet. As advanced cryptography becomes easier to implement and manage, more companies and organizations can take advantage of these benefits.

REFERENCES

[1] Mihir Bellare, Ran Canetti, Hugo Krawczyk, "Message Authentication Using Hash Functions The MAC Construction", *RSA Laboratories CryptoBytes*, Vol.2, No.1, Spring 1996.

[2] Mihir Bellare, Ran Canetti, Hugo Krawczyk , "Keying Hash Functions for Message Authentication", *Advances in Cryptology Crypto 96 Proceedings,* Lecturer Notes in Computer Science Vol. 1109, N. Koblitc ed. Springer Verlag, 1996.

[3] Raphabel C, W.Phan, Aavid Wanger, *Journal of Computers & Security* 25(2006), pp 131-136.

[4] William Stallings, *Cryptography and Network Security: Principles and Practices*, Third Edition, Prentice Hall, 2003.

[5] Ivan Damgard, "A design principle for hash functions", *InAdvances in Cryptology CRYPTO '89* Volume 435 of Lecturer Notes in Computer Science, Pages 416-427, Berlin, NewYork, Tokyo, 1990, Springer Verlag.

[6] Ralph C. Merkle, "One way hash functions and DES", *InAdvances in Cryptology CRYPTO '89* Volume 435 of Lecturer Notes in Computer Science, Pages 428-446, Berlin, New York, Tokyo, 1990, Springer Verlag.

[7] Mihir Bellare, Roch Guerin, Philip Rogeway, "XOR MACs: New Methods for Message Authentication using Finite Pseudorandom Functions", *Advances in Cryptology Crypto 95 Proceedings*, Lecturer Notes in Computer Science Vol. 963, D. Coppersmith ed. Springer Verlag, 1995.

[8] Sheena Mathew, K. Paulose Jacob, "A New Fast Stream Cipher: MAJE4", *Proceedings of IEEE, INDICON 2005,* pp 60-63.

[9] D.E.Knuth, *The Art of Computer Programming* Vol.1., Seminumeical Algorithms, Addision Wlesley, 1969.