National Conference on Emerging Trend & Its Application in Engineering (NCETAE-2011)

An Approach to Data Integrity in Web Applications

Asish Kumar Dalai, Saroj Kumar Panigrahy and Sanjay Kumar Jena
Department of Computer Science and Engineering
National Institute of Technology Rourkela, Odisha, India
dalai.asish@gmail.com, skp.nitrkl@gmail.com. skjena@nitrkl.ac.in

Abstract— **Web application security is extremely important due to the extensive use of web in daily life. Today for almost every purpose we depend on web such as for social networking, ticket booking, online shopping etc. We are exchanging the confidential information as and when needed. Attackers can capture this private information and may modify before being sent to the application server. This paper explains a security model that can be used in web applications to maintain integrity of the data.**

Keywords-Web Security; Message Integrity; Hash Function; Stream cipher

I. INTRODUCTION

Applications on web are highly prone to attack because of its distributed nature. Providing security to these applications is highly essential. Message integrity allows the sender to send a message to the receiver in such a way that if the message is modified in the root, then the receiver will almost certainly detect this. It is needed to protect the integrity of the message ensuring that each message that it is received and deemed acceptable is arriving in the same condition as it was sent out with no bits inserted, missing, or modified. E-commercial sites dealing with financial transactions are susceptible to attacks where attacker tries to modify the parameters which results the loss in data integrity. The proposed model for data integrity ensures that the massage received in the server is same as it was intended to be. Our model ensures that some parameters which must not be modified i.e. server set parameters are sent along with key hashed message authentication code to ensure data integrity.

In this paper we have classified the web application attack into two types and provided the solutions for them

1. **In transit attack:** Attacker is at a remote location intercepts the message and then modifies it before being sent to the server.
2. **Source station attack**: Attacker sitting in the client machine intentionally modifies the parameters and then sends it to the server.

For in transit attack we are using iterated hash function to generate MAC and a MAJE4 stream cipher. For source station attack we apply the hash function to only those parameters which are set by server as fixed parameters. As shown in Fig. 1, the message and the fixed parameters from the server are passed through the iterated hash function to generate message authentication codes for them $H(M)$ and $H(FP)$ respectively. The message M along with the hash codes $H(M)$ and $H(FP)$ are encrypted using MAJE4 with a key K of 128 bit length. The cipher text is then transmitted to the server. Using the same key K and the fast stream cipher MAJE4 the cipher text is decrypted back to produce the message and the hash codes $H(M)$ and $H(FP)$ .Now the server re-computes the hash code of the message and the server set fixed parameters using the same iterated hash function. Hence the server validates the integrity of the message and the integrity of the server set fixed parameters by comparing the hash codes received from client with that generated at the server. If both the codes match then the transmission has done securely.

This paper is structured as follows: Section II describes the working of iterated hash function. Section III contains MAJE4 cipher. Implementation of the security pattern has been described in Section IV Concluding remarks are given in Section V.

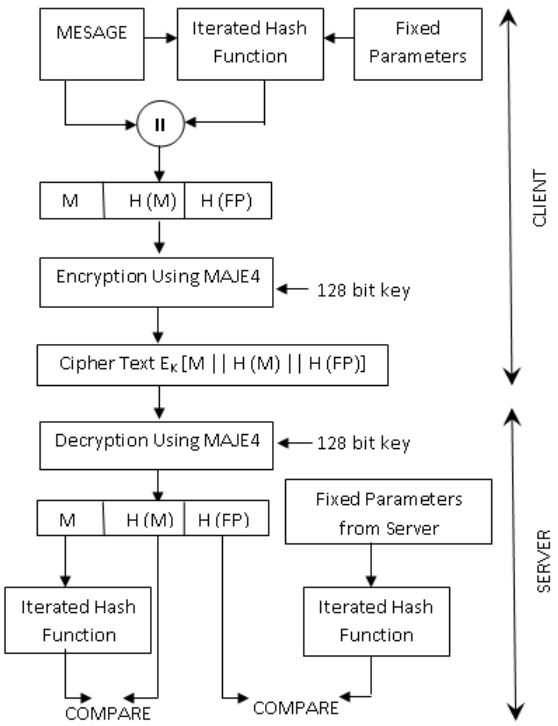

Fig.1. Use of Iterated Hash Function and MAJE4 Encryption

II. ITERATED HASH FUNCTION

We have taken the Merkle-Damgard model for our iterative hash function [9]. 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 numbers of zero 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. $M = M[1].....M[n]$ 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 generate the initial hash code. Then $F2$ is applied again to the initial hash code and to $MB[2]$ to generate 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 generate the final hash code $H(M)$ and $H(FP)$ of 64 bit length.
5. Now $H(M)$ and $H(FP)$ is added with M as the authentication tag.

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) and (2) is random and hence the attackers will not be able to predict the initial value easily. The functions $F1$ and $F2$ in (1) and (2) are ADD and XOR operations [15] 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.

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.

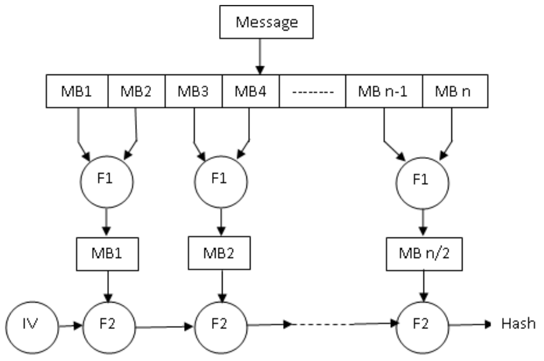

Fig. 2. Model of Iterated Hash Function

That is $H(M) = F2K(F1(M))$ (1)

$H(FP) = F2K(F1(FP))$ (2)

III. MAJE4: A FAST STREAM CIPHER

The 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 [11]. 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 of any length. The algorithm for 128bit MAJE4 cipher is given below.

```
Step 1:   Take key $(K)$ of length 128bit. And $D = 4$
Step 2:   Consider two least significant bits of $K_{[0]}$ and find
          its decimal equivalent and store in the variable ‘ $N$ ’.
Step 3:   $ran = K_{[0]} \wedge K_{[N]}$.
Step 4:   Consider two least significant bits of ran and find its
          decimal equivalent and store in the variable ‘M’.
Step 5:   Check the 16th bit in ran,
          If it is 1 then
          $newran = (K_{[M]} + K_{[M+1 \bmod D]}) \wedge (K_{[M+2 \bmod D]} + K_{[M+3 \bmod D]})$
          Else
          $newran = (K_{[M]} \wedge K_{[M+1 \bmod D]}) + (K_{[M+2 \bmod D]} \wedge K_{[M+3 \bmod D]})$
Step 6:   The output 32-bit word is newran, which can be
          used to $XOR$ with the corresponding word in the
          plain text.
Step 7:   Advance all the keys as
          $K_{[i]} = K_{[i]} * K_{[i]} + K_{[i]} >> 20$
```

Step 8: Go to step2

IV. Using the Model

The following steps are performed to check the integrity of the data using iterated hash function and MAJE4 stream cipher.

1. At the client side hash code of message M and the hash code of fixed parameters are generated by using iterated hash function.
2. Then the message M , hash code of the message $H(M)$ and hash code of fixed parameters $H(FP)$ are encrypted using 128 bit key and the fast stream cipher MAJE4 and sent to the server.
3. The server decrypts the whole message using the same 128 bit key and MAJE4 stream cipher to extract the M , $H(M)$ and $H(FP)$
4. Server re-computes the hash code $H(M)$ and $H(FP)$ over the message M and the server set fixed parameters respectively and then and checks whether they matches with the received hash codes.
5. If $H(M) = F2K(F1(M))$ then the message is considered in transit attack proof and if $H(FP) = F2K(F1(FP))$ then there is no tampering of data at the source i.e. the message is source station attack proof. After these two tests only we can say that the message has reached securely.

The time taken for producing the hash codes using the iterated hash function for the message M and the fixed parameter FP and encryption decryption using the iterated hash function and MAJE4 is given in table I.

TABLE I. Total time taken for producing the hash codes and encryption decryption using the iterated hash function and MAJE4

Plain text (M, FP) (bytes)		Time taken for producing hash codes (Sec.)		Total time taken for encryption and decryption(Sec.)
M	***FP***	***H*(*M*)**	***H*(*FP*)**	$D_K[E_K[M \parallel H(M) \parallel H(FP)]]$
8089	1997	.013	.003	.063
98446	2042	.014	.003	.075
175150	2600	.025	.004	.093
205161	2941	.030	.004	.131
267395	4843	.039	.007	.195

The deterministic random bit generator data for the MAJE4 is given in Table .II

TABLE II. Timing analysis & memory requirement

DRBG	MAJE4
Key length	128-bit
No. of random numbers Generated	1,15,39,399
No. of random bits per each random number	32
Total no. of bits produced (speed Mbps)	352.15
Memory requirement (Bytes**)**	5435

V. Conclusion

From table I and II we came to the conclusion that integrity of the data can be achieved by using a little effort and time by using the proposed model. The additional memory requirement is also not an issue as memory is getting cheaper by these days. It is faster and can be easily implemented by web applications. The additional use of iterated hash function makes the system more reliable by preventing the source station attack. Digital transmissions and online transactions over web can take the advantage of the model.

VI. Acknowledgment

The authors are indebted to Information Security Education and Awareness (ISEA) Project, Ministry of Communication and Information Technology, Department of Information Technology, Govt. of India, for sponsoring this research and development activity.

References

[1] Wikipedia: Transport layer Security, http://en.wikipedia.org /wiki/ Secure_Sockets_Layer.

[2] Sheena Mathew, K.Paulose Jacob, “A New Fast Stream Cipher: MAJE4”, Proceedings of IEEE, INDICON 2005, pp60-63, 2005.

[3] National Institute of Standards and Technology (NIST) FIPS- 180-2: Secure Hash Standard, at http://csrc.nist.gov/publications/fips/fips 180-2/fips 180-2.pdf. 2002.

[4] Mihir Bellare, Ran Canetti, Hugo Krawczyk, “Keying Hash Functions for Message Authentication”, Advances in Cryptology- CRYPTO, LNCS 1109, Springer- Verlag, pp 1-15. 1996.

[5] Mihir Bellare, Ran Canetti, Hugo Krawczyk, “Message Authentication using Hash Functions the HMAC Construction”, CryptoBytes, Vol 2, No.1, RSA Laboratories pp 1-5. 1996.

[6] Thomas Calabrese, “Information Security Intelligence Cryptographic Principles and Applications”, Thomson Delmar Learning, India. 2006

[7] William Stallings, Cryptography and Network Security: Principles and Practices, Fifth Edition, Prentice Hall, 2010.

[8] Ivan Damgard, “A design principle for hash functions”, In Advances in Cryptology - CRYPTO '89 Volume 435 of Lecturer Notes in Computer Science, Pages 416-427, Berlin, NewYork, Tokyo, Springer - Verlag. 1990.

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

[10] Sheena Mathew, K. Paulose Jacob, “Message Integrity in the World Wide Web: Use of Nested Hash Function and a Fast Stream Cipher” - International Conference on Advanced Computing and Communications, 2006. IEEE Conferences, ADCOM 2006.

[11] D.E.Knuth, The Art of Computer Programming, Vol.2, Seminumerical Algorithms, Third Edition, Addison – Wesley, 1997.

[12] Sheena Mathew, K. Paulose Jacob, “Use of Novel Algorithms MAJE4 and MACJER-320 for Achieving Confidentiality and Message Authentication in SSL & TLS”. -Page(s): 444 – 450, World Academy of Science, Engineering and Technology 39, 2008.

[13] Stefan Lucks, “Design Principles for Iterated Hash Functions” e-print (September 29, 2004) http://th.informatik.uni-mannheim.de/people/lucks/

[14] Antoine Joux, “Multicollisions in Iterated Hash Functions. Application to Cascaded Constructions”CRYPTO 2004, LNCS 3152, pp. 306–316, 2004.

[15] Mihir Bellare, etal."XOR MACs: New Methods for Message Authentication using Finite PseudorandomFunctions", Advances in Cryptology - Crypto 95 Proceedings, Lecturer Notes in Computer Science Vol. 963, D. Coppersmith ed. Springer -Verlag, 1995.