SECURITY IN CLOUD COMPUTING USING HASH CODE

Afsaruddin Ahamad[1], Amit Asthana[2], Manik Chandra Pandey[3]

[1,2,3] Department Of Computer Science & Engineering,

Swami Vivekanand Subharti University,Meerut (India)

ABSTRACT

The Hash functions describe as a phenomenon of information security and are used in numerous security applications and protocols such as: - Digital signature schemes, construction of MAC and random number generation, For correct data integrity and data origin authentication. Analysts have find out serious security flaws and vulnerabilities in most widely used MD and SHA family hash functions. As a result hash functions from FORK family with longer digest value were considered as good alternatives for MD5 and SHA-1, but recent attacks against these hash functions have highlighted their weaknesses. In this paper we propose a dedicated hash function HMACMD5= MD5 (HMAC (K,m))//32 BIT based on the design principle of HMAC and MD5. It takes 512 bit message blocks and generates 32 bit hash value. A random sequence is added as an additional input to the compression function of HMACMD5. Three branch parallel structure and secure compression function make HMACMD5 an efficient, fast and secure hash function. Various simulation results indicate that HMACMD5 is immune to common cryptanalytic attacks and faster than NewFORK-256.

Keywords: Hash Function, Provable Security, Quasi-Cyclic Codes, Computer Security, Cryptography, HMAC, MAC, Message Authentication, SHA, MD5 Information Processing Standard (FIPS)

I. INTRODUCTION

Reliable privacy and preventing integrity of a message are the main objectives of cryptography/Encryption. Cryptographic hash functions dropped most important important cryptographic primitives, and they can be used to ensure the security of many cryptographic applications and protocols such as: Digital signature, random number generation, data source authentication, key update and derivation, message authentication code, integrity protection, malicious code recognition and SSL. Hash functions basically compress an input message of fixed length to an output with short fixed length, the hash code.

Hash functions are categorized in to two categories:

1. Unkeyed hash function also known as Manipulation Detection Code (MDC) with single parameter.
2. A message and keyed hash function with two distinct inputs – a message and secret key.

Keyed hash functions are used to construct the MAC (Message Authentication Code). The MAC is widely used to provide data integrity and data origin authentication. The choice between a MAC and an MDC is application may vary. They are also classified as, namely hash functions based on block cipher, hash functions based on modular algorithm and dedicated hash functions. The push for block cipher based schemes is the minimization of the effort to design and implement the hash functions and of the complexity of the equipment. Also the trust that one has in a certain block cipher (such as DES) can be transferred to a hash function. Particularly interesting from a theoretical point of view, modular algorithm based schemes are provably secure, in the sense that their security relies on the hardness of some mathematical problems such as number theory problems. Dedicated hash functions are specially designed from the scratch for the purpose of hashing a plain text with optimized performance and without being constrained to reusing existing system components such as block ciphers and modular arithmetic. These hash functions are not based on hard problems such as factorization and discrete logarithms. The most popular method of designing compression functions of dedicated hash functions is a serial successive iteration of a small step functions.

Our proposed hash function comes under the last category.

a) Preimage resistance,

b) second preimage resistance and

c) collision resistance

are the three classical requirements for security of keyless hash functions. Properties pre image resistance, second pre image resistance and collision resistance are also known as one way, weak collision resistance and strong collision resistance respectively. First of all, it needs to be computationally infeasible to find the pre image X of H(X), when H(X) is given. This is called pre image resistance. Secondly, finding Y ≠ X with H(Y) = H(X), when X andH(X) are given, should also be infeasible. This property is called second pre image resistance. Finally, it should be computationally infeasible to find any two distinct messages X and Y with H(X) = H(Y). This is called collision resistance. Additionally, a hash function is often required to behave indistinguishably from a random function. An ideal hash function that generates an n bit hash value requires evaluating about 2n/2messages to find any pair of messages having the same hash value. This kind of brute-force search for hash functions is called a birthday attack. Also 2n hash computations are required for finding pre images.

In this paper we have proposed a dedicated hash function HMACMD5 that takes a message of arbitrary length and converts it into a 32 bit hash value. It is based on the design principle of NewFORK-256. The compression function of HMACMD5 consists of three parallel branches; each branch compresses a sixteen 32 bit words to eight 32 bit words output. Each branch contains eight step operations. FORK-256 and NewFORK-256, both hash functions built on Merkle-Damgård method. Past few years ago, many attacks have been occurred for these hash functions. The compression function of the proposed hash function uses dithering design. Its padding and splitting of message is similar to Merkle-Damgård construction. Dithering construction is obtained by providing an additional input to the Merkle-Damgård construction. This design was given by Rivest. The loop structure of this design provides good resistance against some crucial attacks such as birthday attack, meet-in-the middle attack and pre image attack. The compression function of HMACMD5 has two input parameters and one output parameter. The additional input parameter is a dither value. There are many ways to select dither value. Random numbers generated from Park–Miller algorithm are used as dither value in the proposed algorithm. For each message block in the message, there is different dither value sequence. For each message block 16 different 32 bit dither value generated, out of which each branch makes the use of eight 32 bit dither value according to the ordering rule. All the modifications made to overcome the recent attacks on FORK-256 and NewFORK-256.

II. PROPOSED ENCRYPTION METHOD

HASHCODE (K,m) = H((K ⊕ opad) ‖ H((K ⊕ ipad) ‖ m))

HASHCODEMD5= MD5(HASHCODE (K,m))//32 BIT

Where,

- H is a cryptographic hash function,
- K is a secret key padded to the right with extra zeros to the input block size of the hash function, or the hash of the original key if it's longer than that block size,
- m is the message to be authenticated,
- ‖ denotes concatenation,
- ⊕ denotes exclusive or (XOR),
- opad is the outer padding (0x5c5c5c…5c5c, one-block-long hexadecimal constant),
- and ipad is the inner padding (0x363636…3636, one-block-long hexadecimal constant).

III. PSEUDOCODE OF ALGORITHM

The following pseudocode demonstrates how HASHCODE may be implemented. Blocksize is 64 (bytes) when using one of the following hash functions: SHA-1, MD5, RIPEMD-128/160.

functionString HashCode (key, message)

if (length(key) > blocksize) then

key = hash(key) // keys longer than blocksize are shortened

end if

if (length(key) < blocksize) then

key = key ‖ [0x00 * (blocksize - length(key))] // keys shorter than blocksize are zero-padded (where ‖ is concatenation)

end if

o_key_pad = [0x5c * blocksize] ⊕ key // Where blocksize is that of the underlying hash function

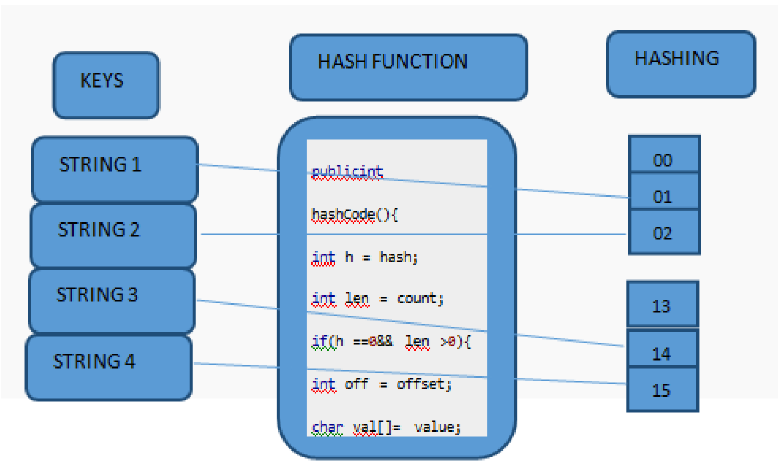
i_key_pad = [0x36 * blocksize] ⊕ key // Where ⊕ is exclusive or (XOR)

return hash(o_key_pad ‖ hash(i_key_pad ‖ message)) // Where ‖ is concatenation //proposed algo apply HASH Function on Returned value with 32 bit int Variable

end function

String 32 VAL = md5(HashCode (key,msg))

IV PERFECT HASHING



A hash function that is injective—that is, maps each valid input to a different hash value—is said to be perfect. With such a function one can directly locate the desired entry in a hash table, without any additional searching.

V. CONCLUSION

I Proposed hash function generates 32 bit hash string. Proposed a dedicated hash function HMACMD5= MD5 (HMAC (K,m))//32 BIT based on the design principle of HMAC and MD5. It takes 512 bit message blocks and generates 32 bit hash value. A random sequence is added as an additional input to the compression function of HMACMD5. It has a parallel structure consist of three branches. Each branch computes eight step operations. Algorithm uses 16 constants and two nonlinear functions. Each branch makes the use of 16 message sub blocks and constants with different order. For making the whole structure complicated for the cryptanalyst and secure against known attacks compression function uses three inputs. Dither values are used as third input. These dither values are random numbers generated through Park–Miller algorithm. The one way structure of the algorithm makes it strong against preimage and second preimage attack. Various tests have been performed to check the security level of hash function. The bit variance test has been performed for one bit changes. Simulation results and rigorous analysis of the hash function guarantee its security against differential and other common known attacks. Further, as future work, I will try to improve its efficiency.

VI. ACKNOWLEDGMENT

I would like to give this work to my parent, who have created the author of this paper. I would like to thank Assistant to the Professor Mr. my advisor AMIT ASTHANA , for his understanding and research guidance. He has always been the one who guided me through difficulties final completion of my paper. The idea of Hash Code Based Security In Cloud Computing is in fact due to him and to the department staffs who has helped to create a computing environment that enabled us to work effectively.

REFERENCES

[1] Alfred M., Oorschot P., and Vanstone S., Handbook of Applied Cryptography, CRC press, 1997.

[2] Bruce S., Applied Cryptography: Protocols, Algorithms and Source Code in C, John Wiley and Sons, Canada, International Journal of Advanced Technology in Engineering and Science www.ijates.com Volume No.02, Issue No. 06, June 2014 ISSN (online): 2348 – 7550 Page | 41 1996.

[3] Stallings W., Cryptography and Network Security Principles and Practices, Prentice Hall Press Upper Saddle River, 2010.

[4] Goldwasser S., Micali S., and Rivest R., “A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks,” Journal on Computing, vol. 17, no. 2, pp. 281-308, 1988.

[5] Ilya M., “Hash Functions: Theory, Attacks, and Applications,” in Proceedings of Microsoft Research, Silicon Valley Campus, pp. 1-22, 2012.

[6] National Institute of Science and Technology, “Secure Hash Standard,” Federal Information Processing Standard 180-1, available at: http:/ http://www.itl.nist.gov/fipspubs/fip180-1.htm, last visited 1995.

[7] National Institute of Science and Technology, "Secure Hash Standard," Federal Information Processing Standard 180-2, available at: http://www.itl.nist.gov/fipspubs/fip180-1.htm, last visited 2002.

[8] National Institute of Standards and Technology, "Secure Hash Standard," "FIPSPUB180 www.itl.nist.gov/fipspub/fips180-1.html, last visited 2003.

[9] National Institute of Science and Technology, "Implementing Cryptography," NIST SP 800, available at: http://csrc.nist.gov/publications/ nistpubs/800-21-1/sp800-21-1_Dec2012.pdf, last visited 2012.

[10] "New European Schemes for Signatures, Integrity and Encryption Project," available at: http://www.cryptonessie.org, last visited 2000.

[11] Phan R. and Wagner D., "Security Consideration for Incremental Hash Function Based on Pair Blocking Chaining," in Proceedings of Computers and Security, USA, pp. 131-136, 2006.

[12] Rivest R., Shamir A., and Adleman L., "A Method for Obtaining Digital Signature and Public-Key Cryptosystems," Communication of The ACM, vol. 21, no. 2, pp. 120-126, 1978.

[13] Sklavos N., Alexopoulos E., and Koufopavlou O., "Networking Data Integrity: High Speed Architectures and Hardware Implementations," The International Arab Journal of Information Technology, vol. 1, no. 0, pp. 54-59, 2003.

[14] United States Department of Commerce, National Bureau of Standards, Data Encryption Standard, Federal Information Processing Standards Publication, 1977.

[15] William S., Cryptography and Network Security, Principles and Practice, Prentice Hall of India, 2012.