iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 6 **Issue:** IV **Month of publication:** April 2018

DOI: http://doi.org/10.22214/ijraset.2018.4240

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 6.887
Volume 6 Issue IV, April 2018- Available at www.ijraset.com

An Enhanced Hash-based Message Authentication Code using BCrypt

Jerone B. Alimpia[1], Dr. Ariel M. Sison[2], and Dr. Ruji P. Medina[3]
[1, 2, 3] *Technological Institute of the Philippines - Quezon City, Philippines*

Abstract: ***Many message authentication codes like HMAC depend on its underlying cryptographic algorithm. Unfortunately most of these algorithms are very fast that permits the attacker to easily establish a brute-force attack for key-retrieval. In this paper, it presents a method of solving the issue by applying BCrypt Expensive Key Setup function to derive the secret key of HMAC. As a result, the modification helps the algorithm to strengthen its resistance to exhaustive search and it protects against key recovery attacks. This work also presents results of comparing time-complexity in performing key-recovery attack for both original and modified version of HMAC.***
Keywords: HMAC, MAC tag, key-recovery, hashing, bcrypt, brute-force

I. INTRODUCTION

In cryptography, one-way hashing was a ground breaking method to protect information against different kinds of threat like brute-force attack [1]. In the same manner, cryptographic hashing plays a very significant role for many message authentication process [2]. Ever since, verifying message is a prime necessity in computer systems and networks, thus message authentication algorithm like hash-based message authentication code (HMAC) is important to provide integrity check and authentication particularly when a message travels over unsecure network[3]. Primary job of HMAC is relatively similar to most MAC algorithm like CBC-MAC (Cipher Block Chaining), UMAC, VMAC and etc. HMAC uses cryptographic hash function (e.g. MD5 or SHA-1) to produce a MAC tag by condensing a secret key (only known by sender and receiver) and message as input into HMAC algorithm. This MAC tag is just normally a hash value used for authenticating the message. After producing the MAC tag, it is typically send to the intended receiver along with the message, and then the receiver will also produce another MAC tag using the received message and secret key into the same HMAC function as were used by the sender. If the first MAC tag is identical to the second MAC Tag, the receiver can safely assume that the message was not altered or tampered during transmission [4].

However, after long years of common practice and fast growing hardware improvements, HMAC's associated weaknesses and vulnerabilities have been identified (citations will be discuss below). Since HMAC uses hashing technique to generate the MAC tag [5], one practical way on how to challenge this algorithm is through Brute-force[6]. Brute-force is a trial and error method used by application programs to decode encrypted data like MAC tag [7]. Using computer hardware with tremendous computational speed and power, the attacker can simply generate a large number of random keys with the message and run it with through HMAC function until the attacker finds the right key that generates the same MAC tag. This common technique is used against HMAC algorithm to retrieve the key secret from specific a MAC Tag [5]. The biggest problem when the attacker recovers the key (key recovery) the attacker can use it to generate a message and disguise as the real sender of the message without any knowledge of the true receiver (forgery). But aside from this well-known attack, many devastating attacks are being reported such as the full key-recovery attacks by[8], forgery attacks by[9], key recovery attacks by[10], and the latest near-collisions attack presented by [11]. Moreover, given that hardware improvements constantly give attackers increasing computational power. And as microprocessor grows faster, variety of attacks are widely used with cleverly optimized implementations that can give opportunity for well-funded adversary to achieve their detrimental goal [12]. Furthermore, with respect to Moore's law, as CPU's performance continues to increase, hash based algorithms like HMAC will continue to become weaker [13].

The above mentioned attacks totally undermined the confidence of HMAC algorithm specifically brute-force. Fortunately there is certain algorithm designed to deter a kind of attack like brute-force even with increasing computation power. This algorithm popularly known as BCrypt firstly presented by Provos and Mazieres in USENIX. BCrypt is one of the most powerful algorithms which are quite successful in restraining unwanted attacks in the system[14]. It was designed to be an adaptive function which made it resistant to brute force attacks and remain secure despite of hardware improvements [15].

Thus, this study will modify the original HMAC algorithm by (a) adapting the same properties used by BCrypt which is the ExpensiveKeySetup function, (b) to add another input parameter that will dictate the amount of work factor exponentially in

©IJRASET (UGC Approved Journal): All Rights are Reserved

generating the MAC Tag, (c) to employ the Modular Crypt Format standard and lastly (d) to measure the security of the proposed modified version of HMAC compare to the original version in resistance to brute-force and key recovery attack even against powerful attacker and future hardware improvements.

II. RELATED LITERATURE

Hashing plays a very significant role in information security regarding passwords [2] but nowadays it is also revolve for ensuring data integrity and authentication. In computer science, hashing is the process of taking a sequence of characters, known as the plaintext, and then transforming that to a usually shorter fixed-length sequence of characters, known as the hash value. A hashing algorithm converts a variable length message into a condensed representation of the electronic data in the message and often utilized with digital signature algorithms, key derivation functions, random number generators and keyed-hash message authentication codes [16]. The foundation for most of cryptographic algorithms is built upon this concept. Since verifying data is a prime necessity in computer systems and networks, there's certain algorithm that main purpose is to protect both data integrity as well as authenticity of information when it travels over unsecure channel or medium, this mechanism known as Message Authentication Code or MAC. This algorithm uses secret key that is usually shared between two parties; the sender of message and the verifier who authenticate the message. In this case, the sender put the message through a MAC algorithm to generate the authentication tag or also known as the MAC tag and send it to the receiver together with the key, this key is a secret key that is shared between the sender and the intended receiver(s)[16]. The verifier also put the message through the same MAC algorithm using the shared secret key, producing a second MAC data tag.

The verifier then compares the first MAC tag received in the transmission to the second generated MAC tag. Then if both MAC tag are identical, the receiver can safely assume that the message was not altered or tampered with during transmission. More precisely, the MAC algorithm protects both a message's data integrity as well as its authenticity, by allowing verifiers to detect any changes to the message content or any types of message forgery. Until today, MAC is one of the most important and widely used cryptographic tools. In literature there have been two mainly types of MAC algorithms, first, the block cipher based MAC algorithms and second, the hash function based MAC algorithms.

Primarily MAC's was typically underlying out of block ciphers. A block cipher is a method of encrypting text to produce cipher text in which a cryptographic key and algorithm are applied to a block of data. CBC-based MACs is the most common MAC algorithm based on a block cipher makes use of cipher block-chaining in which a sequence of bits are encrypted as a single unit or block with a cipher key applied to the entire block [17]. The CBC MAC is an international standard [18]. This standard is extensively employed in the banking sector and in other commercial sectors. However, since there are many cryptographic schemes, na¨ıve use of ciphers and other protocols may lead to attacks being possible, reducing the effectiveness of the cryptographic protection what makes the CBC MAC unsecure [19]. Until eventually, there has been a rise of interest in the idea of constructing new MAC's algorithm out of cryptographic hash functions. And that's the reason why the original construction for HMAC was firstly introduced. HMAC defined as Hash-based Message Authentication Code presented by Mihir Bellare, Ran Canettiy and Hugo Krawczyk in February 1996. HMAC is an explicit kind of MAC also used for the assurance of data integrity and authentication [3]. Unlike CBC-MAC, HMAC does not encrypt the message, instead, the message must be sent alongside the HMAC algorithm using any cryptographic hash function. After receiving a recommendation and approval from FIPS since July 2008, until now HMAC is standardized (by ANSI, IETF, ISO and NIST) and widely deployed (e.g. SSL, TLS, SSH, Ipsec) [20].

HMAC has the property to use any cryptographic hash function as internal component which processes short fixed-length inputs, and then iterated in a particular way in order to hash arbitrarily long inputs. For simplicity let's assume H to be the SHA-1 hash function where data is hashed by iterating a basic compression function on blocks of data. While B denote the byte-length of such blocks, and by L the length of hash outputs. The authentication key K can be of any length up to B, the block length of the hash function. Keys longer than B bytes will first hash the key using H and then use the resultant L byte string as the actual key to HMAC. In any case the minimal recommended length for K is L bytes (as the hash output length). The two fixed and different strings ipad and opad define as follows (the 'i' and 'o' are mnemonics for inner and outer):

ipad = the byte 0x36 repeated B times

opad = the byte 0x5C repeated B times.

To compute HMAC Tag over the data "text" this method need to perform:

MAC Tag = H ((K'⊕ opad)|| H((K'⊕ ipad) || text))

©IJRASET (UGC Approved Journal): All Rights are Reserved

Providing a way to check the integrity and authenticity of information specially for message that are transmitted even over unreliable network, it is a prime necessity for HMAC to have a strong underlying cryptographic hash function since the strength of this algorithm depends upon the cryptographic strength of its underlying hash function. According to its original authors, HMAC can be used with any cryptographic hash function as long as it has a property to break up a message into blocks of a fixed size and iterates over them with a compression function. Publicly, FIPS recommendation is to use either MD5 or SHA-1 as cryptographic hash function for internal component as standard for HMAC algorithm [21]. However after a long years of common practice and implementation, associated weaknesses and vulnerabilities have been identified suggesting that these two algorithms might not be secure enough anymore for ongoing use and implementation[22]. In year 2006 Jongsung Kim, Alex Biryukov, Bart Preneel, and Seokhie Hong presented a paper that shows how to recovery the secret key on HMAC with reduced or full versions of these two cryptographic hash functions from a random function. A differential distinguisher is a technique use to allow attacker to devise a forgery attack on HMAC. The goal of this kind of attack is to uncover the secret key of specific MAC tag and use it to establish forgery with any message they want. In cryptography, forgery is the sending of a message to deceive the recipient as to whom the real sender is. Furthermore as a result of their work, they achieved to measure the vulnerabilities of the two hash functions carry over to the HMAC construction. Three years later strong related attack also reported in 2009 when Xiaoyun, Hongbo, Wei, Haina, & Tao presented a key recovery attack on HMAC-MD5 without using related keys. It can distinguish an instantiation of HMAC with MD5 from an instantiation with a random function with 297 queries with probability 0.87. But aside from distinguishing attack, full key recovery attacks are also reportedly the strongest attack against HMAC constructions.

III. DESIGN ARCHITECTURE

The figure below shows the modified version of HMAC producing a MAC tag over the data 'message' and derived 'key'. In this example, the sender put the message and the derived secret key through the HMAC algorithm to produce a MAC data tag. The message and the MAC tag are then sent to the receiver. The receiver in turn runs the message portion of the transmission through the same HMAC algorithm using the same key, producing a second MAC data tag.

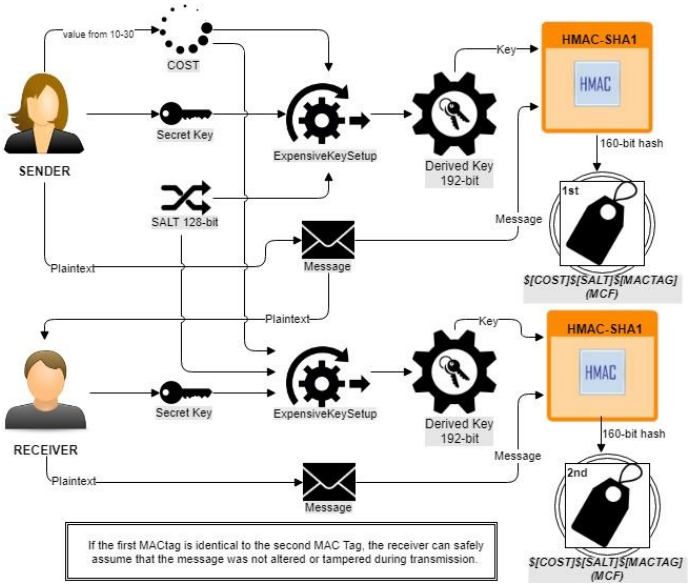


Figure 1. Modified HMAC Diagram

The receiver then compares the first MAC tag received in the transmission to the second generated MAC tag. If they are identical, the receiver can safely assume that the message was not altered or tampered during transmission.

A. Adding Cost Parameter

The cost parameter determines how expensive and how long to compute the secret key. The increase in the cost factor is exponential (as 2cost factor), meaning the higher the COST value, the more expensive the key will be. Given this example, if COST=10 then it requires 210 or 1024 iterations of data encryption to derive the secret key.

©IJRASET (UGC Approved Journal): All Rights are Reserved

B. Applying the Modular Crypt Format (MCF) Standard

Table I. New MAC Tag Format

Cost	Salt	MAC Tag
$10	$545B6768B95F8C3A 90BE3A951CA771F9	$62D03CE951798F864A0C 490B97A40F9DF726A97A

A number of the hashes are described as adhering to the "Modular Crypt Format". This is an attempt to employ the MCF standard for the encoding of the modified HMAC Tag. Instead of using the result of SHA-1 as a MAC Tag, the modular crypt format (MCF) will be used to encode the hash strings of the MAC Tag. The format of the MAC Tag will be encoded as ${cost}${salt}${mactag} where the output MAC Tag should start with ${cost} which can be any number provided by the sender that will dictate the work factor for processing the MAC tag, followed by ${salt} which is a 128bit random string and lastly the ${mactag} which depend on what kind of hash function was being used.

C. Key Derivation Phase

The proposed method of deriving the input key will be performed by adapting the ExpensiveKeySetup function of BCrypt. This function will be called using the cost, salt, and key as inputs to produce a new 192-bit derived key. Salt is a 128-bit value that is randomly generated using java Secure Random class. The primary function of salt is to defend against dictionary attacks or against its hashed equivalent, a pre-computed rainbow table attack. The following pseudocode demonstrates how the modified HMAC was implemented using ExpensiveKeySetup [23]:

Algorithm 1 Modified HMAC

```
Input: key, message, salt, cost, blockSize
Output: mactag
Funtion: hash, ExpensiveKeySetup, Padding
Initialisation:
1 :  dKey ←ExpensiveKeySetup(key,salt,cost)
2 :  if (dKey.length > blockSize) then
3 :     dKey←hash(dKey)
4 :  end if
5 :  if (dKey < blockSize) then
6 :     dKey←Padding(dKey, blockSize)
7 :  end if
8 :  opad = DerivedKey ⊕ [0x5c * blockSize]
9 :  ipad = DerivedKey ⊕ [0x36 * blockSize]
10: return $cost || $salt || hash( opad || hash( ipad || msg ) )
```

Algorithm 2 ExpensiveKeySetup

```
Input: key, salt, cost
Output: dKey
Initialisation:
1 :  state ← InitState ()
2 :  state ← ExpandKey (state, salt, key)
3 :  repeat (2^cost)
4 :     state ← ExpandKey(state, salt)
5 :     state ← ExpandKey(state, key)
6 :  return dKey
```

©IJRASET (UGC Approved Journal): All Rights are Reserved

Where:

hash	is a cryptographic hash function (e.g. SHA-1)
Padding	pad key with zeros in blockSize bytes long
key	is the secret key
msg	is the message to be authenticated
salt	128-bit random string
cost	work factor (e.g. 10-30)
blockSize	the block size of the underlying hash function (e.g. 64 bytes for SHA-1)
dKey	is the derived key using ExpensiveKeySetup function.
‖	denotes concatenation
⊕	denotes exclusive or (XOR)
opad	is the xored result of DerivedKey and outer padding (0x5c5c5c…5c5c, one-block-long hexadecimal constant)
ipad	is the xored result of DerivedKey and inner padding (0x363636…3636, one-block-long hexadecimal constant)

D. Key Recovery on HMAC

To measure the security of both modified and original version of HMAC, both algorithms need to go under a key-recovery attack. This attack will attempt to recover the secret key out of specific MAC tag. Assuming that the attacker obtained the sender's message and the MAC Tag, in this scenario the attacker will generate a number of random keys using a brute-force search program and run it into HMAC function until the program finds the right key that generates the same MAC Tag. The attack will be considered successful if the test finds the right key that can generate identical MAC Tag. The attack operates as follows;

while ($T^1 \neq T^2$)
  K`← B(K)
  T^2 ← HMAC(M , K`)
end while

The attacker first generates a sequence of key K` by means of using the brute-force function B(K). Then the generated key K` and the obtained message M are both used as input for every HMAC(M,K`) iteration that would generate the MAC tag T^2 and finally compared to the obtained MAC tag T^1 .

E. Measuring Time Complexity

To measure how quickly a brute-force successfully discover the key out of a specific MAC tag, the attack definitely needs to rely on some factors like key length, calculation speed, and total numbers of character set to be able to determine how long it would take to determine the key. To estimate the calculation time, the following formula will be needed:

Est. Time = (No. of Character set ^ Key Length) / HPS

To get the total number of character set from the candidate key, there is a need to check if the key contains lowercase, uppercase, numbers, or special character.

Lowercase Letter (a-z)	26
Uppdercase Letter (A-Z)	26
Numbers (0-9)	10
Special Characters	32

Ex. Consider a typical desktop computer with a speed of 1,000,000HPS, using the formula above the time complexity for retrieving the key "Asd123" is computed as;

No. of Charset = 26 + 26 + 10 = 62

Key Length = 6

©IJRASET (UGC Approved Journal): All Rights are Reserved

HPS = 1,000,000
Est. Time = 62^6 / 1,000,000
= 56,800,235,584 / 1,000,000
= 56,800 / 60*60
= 15.77hrs.

GPU or 3D card can also be used to calculate the time complexity of the attack with a speed around 50-100 times greater than a modern computer.

IV. RESULTS

A. Key Recovery Result using Original HMAC

Section 3.2 shows how to retrieve the secret key from a MAC tag produced by the original HMAC algorithm. Therefore on this experiment, using a Java program, a key recovery attack was performed to prove that the original HMAC is no longer secure in protecting the secret key from a MAC Tag against today's hardware. Simulation using Intel(R) Core(TM) i5-3320M CPU 2.60GHz (4 CPUs) with 20,000 HPS, the secret key from this given MAC tag "5CD706BB919C2623723541109AC97FC9600FB042" and message "hello" is shown in Figure 2. Table I shows result of test using different key spaces with the same HPS.

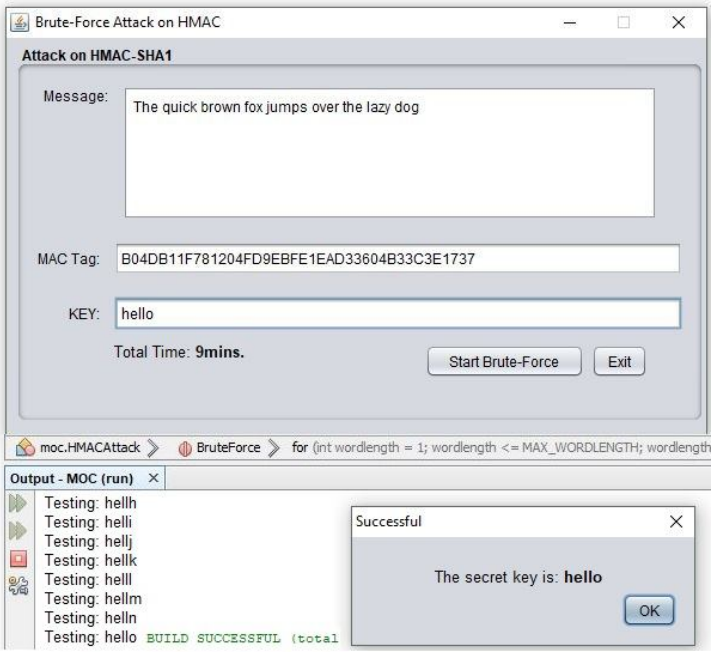


Figure 2. Key Recovery attack via Java Program

By applying the same method on Section 3.5 the key recovery time complexity was measured as:
Approx. Time = (26 ^ 5) / 20,000HPS
= 11,881,376 / 20,000HPS
=594sec. / 60
= 9mins.

Table II. Time Complexity for Original HMAC

Key Space	Possible Combination	Approx. Time
26^5	11,881,376	9.90mins.
36^5	604,66,176	50.38mins.
42^5	130,691,232	1.81hr.
56^5	550,731,776	7.64hr.
68^5	1,453,933,568	20.19hr.

©IJRASET (UGC Approved Journal): All Rights are Reserved

B. Key Recovery Result using Modified HMAC

The HMAC is considered broken if the attacker, successfully retrieve the key out of the obtained Message and MAC Tag. Therefore, to protect the secret key, a key derivation method was implemented (Section 3.3). By applying the modified HMAC, the new format for MAC Tag will become:

MAC Tag	$10$0040103B64EBEB630CB20D04E5DEF6C C$93343BDAC9CAE60F89E11B8CAFF54782 7C9E1129

The new MAC tag will be encoded using Modular Crypt Format. However, there's no official specification document describing this format but it is more organised than the original. Moreover, since the key derivation phase makes the hashing process dramatically slow the value for HPS will also decrease. The purpose of implementing the ExpensiveKeySetup function to derive to secret key is to make each HMAC operation timely expensive to deter brute-force attacks. The first prefix $10 specifies a cost parameter of 10, indicating 2^{10} or 1024 numbers of iteration needed to derived the secret key. While following 128-bit is the random salt $0040103B64EBEB630CB20D04E5DEF6CC and lastly the MAC tag $93343BDAC9CAE60F89E11B8CAFF547827C9E1129.

Table III. Time Complexity for Modified HMAC

Key Space	Possible Combination	HPS using cost of 10	Approx. Time
26^5	11,881,376	12	11.45 days
36^5	60,466,176	12	58.32 days
42^5	130,691,232	12	126.05 days
56^5	550,731,776	12	17.70 mos.
68^5	1,453,933,568	12	3.89 yrs.

Using the same method in Section 3.5 and the input message of "The quick brown fox jumps over the lazy dog", the key recovery time was measured as follow:

Approx. Time = (26 ^ 5) / 12HPS
= 11,881,376 / 12HPS
= 11.45days

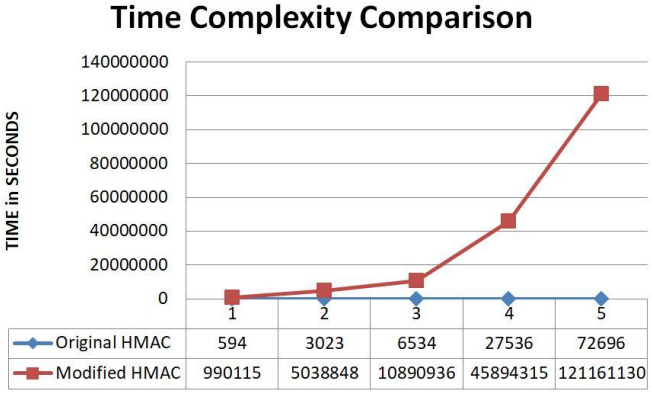


Figure 3. Original vs. Modified HMAC

The figure above demonstrates a ratio of 1:1666 between the original and the modified version of HMAC with respect to time complexity in retrieving the secret key.

V. CONCLUSION AND RECOMMENDATION

xperimental results show that the original HMAC algorithm is no longer secure in protecting the secret key of specific MAC tag against brute-force and key recovery attack. However, this problem was addressed using the ExpensiveKeySetup function. The adaptation of ExpensiveKeySetup on the original HMAC algorithm successfully reduces the hashing process in producing the MAC

©IJRASET (UGC Approved Journal): All Rights are Reserved

tag. Therefore, the modified algorithm has strengthened its security and greatly extends the time complexity making brute-force and key recovery attacks impractical.

Any optimization study can be made on this modified version of HMAC particularly in the setting of parameter values since todays modern CPUs are quite a bit faster. It is also recommended to establish a benchmark to measure how long it would take on the processor to produce a single MAC tag and determine the maximum number of *COST* tolerable and make this algorithm harder to attack.

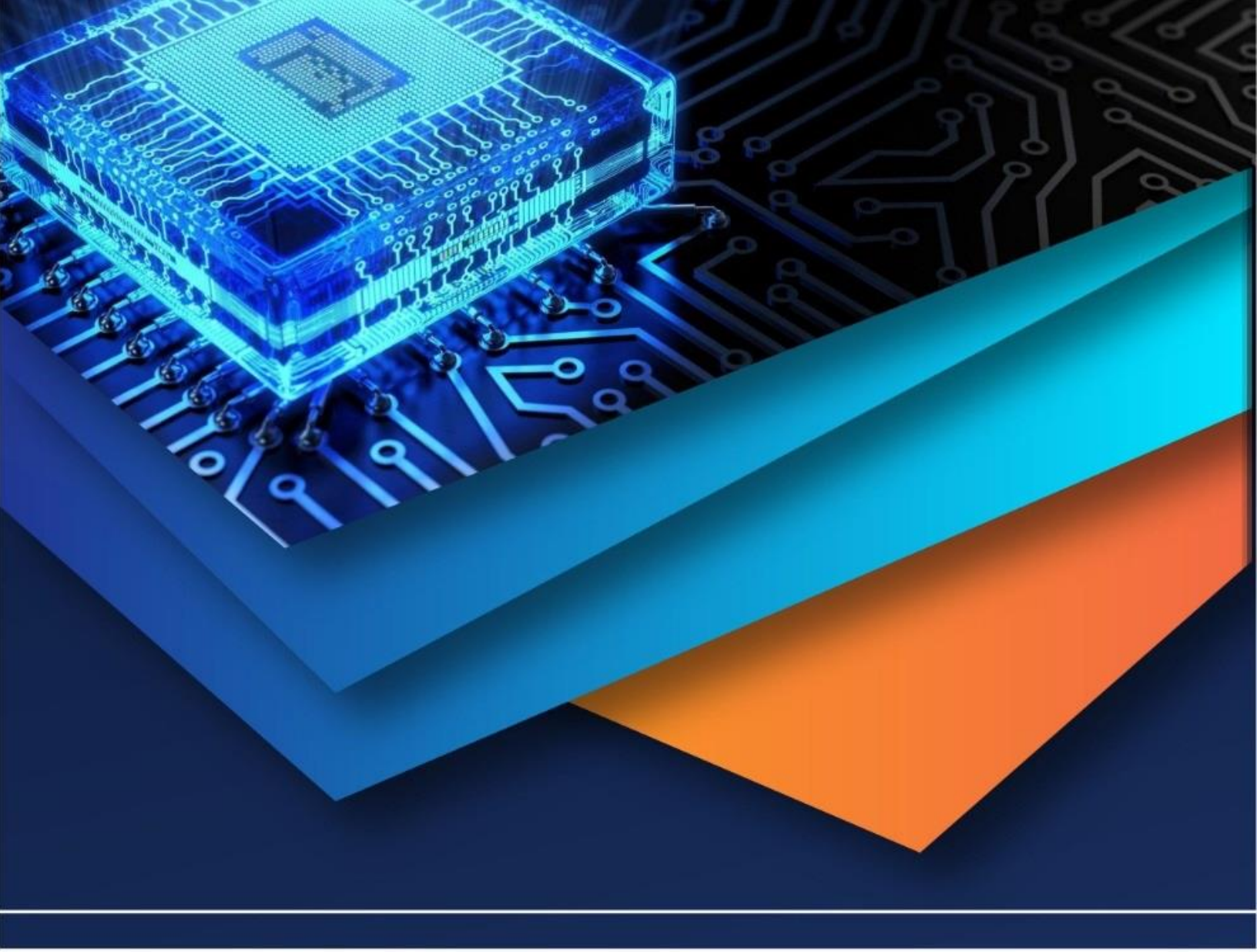
VI. ACKNOWLEDGMENT

It is a pleasure to acknowledge Dr. Cristina Aragon for her important advice, corrections, and suggestions, and for spending their precious time on my research.

REFERENCES

[1] G. Khalil, “Password Security—Thirty-Five Years Later,” SANS Inst. InfoSec Read. Room, 2014.
[2] J. M. Krotkiewicz, “An In-Depth Look into Cryptographic Hashing Algorithms,” 2016
[3] S. Shaker, “HMAC Modification Using New Random Key Generator,” vol. 14, no. 1, pp. 72–82, 2014.
[4] S. J. Samuel and S. J. Jenitha, “Enhanced security and authentication mechanism in cloud transactions using HMAC,” 2014 IEEE Int. Conf. Comput. Intell. Comput. Res. IEEE ICCIC 2014, pp. 0–3, 2015.
[5] D. Ravilla and C. S. R. Putta, “Implementation of HMAC-SHA256 algorithm for hybrid routing protocols in MANETs,” 2015 Int. Conf. Electron. Des. Comput. Networks Autom. Verif. EDCAV 2015, pp. 154–159, 2015
[6] C. Rechberger and V. Rijmen, “New results on NMAC/HMAC when instantiated with popular hash functions,” J. Univers. Comput. Sci., vol. 14, no. 3, pp. 347–376, 2012.
[7] K. Brown, “The Dangers of Weak Hashes,” SANS Inst. InfoSec Read. Room, 2013.
[8] P. Q. Nguyen, P. Fouque, and G. Leurent, “Full Key-Recovery Attacks on HMAC / NMAC-MD4 and To cite this version :,” 2011
[9] T. Peyrin, Y. Sasaki, and L. Wang, “Generic Related-Key Attacks for HMAC,” Asiacrypt, vol. 7658, pp. 580–597, 2012.
[10] J. Guo, Y. Sasaki, L. Wang, and S. Wu, “Cryptanalysis of HMAC / NMAC - Whirlpool,” no. December, pp. 21–40, 2013.
[11] L. Wang, K. Ohta, and N. Kunihiro, “New Key-Recovery Attacks on HMAC/NMAC-MD4 and NMAC-MD5,” vol. 10210, no. October, p. 590, 2017.
[12] M. Dürmuth and T. Kranz, “On password guessing with GPUs and FPGAs,” Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 9393, pp. 19–38, 2015
[13] G. Strawn and C. Strawn, “Moore’s Law at Fifty,” IT Prof., 2015
[14] L. Ertaul, M. Kaur, and V. Gudise, “Implementation and Performance Analysis of PBKDF2, Bcrypt, Scrypt Algorithms,” WORLDCOMP, 2014.
[15] K. Malvoni, “Are Your Passwords Safe : Energy-Efficient Bcrypt Cracking with Low-Cost Parallel Hardware,” no. Algorithm 1, 2014.
[16] Q. H. Dang, “Secure Hash Standard,” FIBS 180-4 Publ., vol. 4, no. August, p. 36, 2015\M. M. Mathews and P. V, “Date time keyed - HMAC,” in 2016 Online International Conference on Green Engineering and Technologies (IC-GET), 2016, pp. 1–5
[17] ISO/IEC, “Mechanisms using a block cipher,” ISO/IEC, vol. 2011, 201
[18] M. Bellare, J. Kilian, and P. Rogaway, “The Security of the Cipher Block Chaining Message Authentication Code,” J. Comput. Syst. Sci., 2000.
[19] P. Fouque, G. Leurent, and P. Q. Nguyen, “Full key-recovery attacks on HMAC/NMAC-MD4 and NMAC-MD5,” Adv. Cryptol. - CRYPTO 2007. Proc. 27th Annu. Int. Cryptol. Conf., pp. 13–30, 2007.
[20] C. Furlani, “The Keyed-Hash Message Authentication Code,” Fed. Inf. Process. Stand. Publ., no. July, 2008.
[21] M. Stevens, E. Bursztein, P. Karpman, A. Albertini, and Y. Markov, “The first collision for full SHA-1,” pp. 1–23, 2017.
[22] N. Provos and D. Mazieres, “A future-adaptable password scheme,” USENIX Annu. Tech. Conf. …, pp. 1–12, 1999.

©IJRASET (UGC Approved Journal): All Rights are Reserved



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)