Novel Approach for Key-Based Hashing Algorithm

Kunj J Joshi[1], Dwireph K Parmar[2], Dhrumin B Patel[3], Kaushal Shah[4]

[1,2,3]Student, Computer Engineering Department, Pandit Deendayal Energy University, India

[4]Assistant Professor,Computer Engineering Department,Pandit Deendayal Energy University, India

Abstract The existing key-based hashing algorithms, such as HMAC, NMAC, and GMAC are highly dependent on pre-existing hashing algorithms such as MD5 and SHA1 which are self-sufficient and independent from the usage of keys. Hence, as a result, the key-based hashing algorithms inherit many aspects of security from their underlying algorithm and usually enhance it. But this feature also accounts for the loopholes in key-based hashing algorithms. The key-based hashing algorithms also inherit the number of rounds, used to hash the message, from its underlying algorithm as well. A fixed number of rounds also allows attacks on particular rounds in the algorithm to crack it. In this paper, a new, independent hashing algorithm is introduced which does not have a fixed number of rounds and is based on fundamental mathematical and Boolean concepts of Bitwise Operators. The performance and security analysis has shown that the proposed algorithm is resistant to known attacks on hashing algorithms and a few other attacks presented by other researchers.

Keywords: Message Authentication Algorithm, Key-based Hashing, Cryptography, Message Passing

1) Introduction

The key-based message authentication algorithms in use today, are usually based on key-based hashing algorithms.[1] The widely used key-based hashing algorithms such as keyed Hash Message Authentication Code (HMAC) [2], keyed Nested Message Authentication Code (NMAC) [3] and Galois-based Message Authentication Code (GMAC) [4], use pre-existing hashing algorithms such as Message Digest 5 (MD5) and Secure Hashing Algorithm 1 (SHA1) as their underlying bases, to calculate a hash value for a given input **Error! Reference source not found.**. They convert the given plaintext into an unreadable form, called hashtext, using the key and the algorithm. Hashtexts generally act as digital fingerprints for any of the hashed materials, whether be it files or text or any material which can be digitally parsed [5].

Algorithms such as SHA512 and SHA256 are considered to be the most secure hashing algorithms in use today, while algorithms such as MD5 and SHA1 have been successfully attacked upon and cracked by researchers to be deemed insecure. The security of a hashing algorithm is based on three different characteristics [6]. These characteristics are as follows:

1) The hashing algorithm must always give a fixed-length output. The output must not vary according to the input.
2) It should be mathematically infeasible to calculate the plaintext from a given hashing algorithm and a corresponding hashtext. (Pre-image Resistance)
3) A hashing algorithm must not provide similar hashtexts for two different plaintexts given to it as its input. (Collision Resistance)

With researchers focusing their energy on these three characteristics, they have been successful in disproving Collision Resistance for highly used hashing algorithms such as MD5 and SHA1. Certain qualities of the algorithm such as fixed number of rounds have given way to researchers to attack specific rounds and exploit the vulnerabilities in the hashing algorithm **Error! Reference source not found.**. These drawbacks are covered up by key-based hashing approach, but hashes such as HMAC and NMAC inherit the qualities of its underlying hashing algorithm and hence have been attacked and successfully cracked by many researchers as well[9].

The work is mainly focused on presenting a novel key-based hashing algorithm that is based on simple Mathematical and Boolean concepts of Bitwise Operators. The novel algorithm overcomes the loophole of existing key-based hashing algorithms, by not being dependent on existing algorithms and defining a new algorithm for converting a plaintext to corresponding hashtext. It overcomes the drawbacks of underlying algorithms such as MD5 and SHA1, by not having a fixed number of rounds to calculate the hashtext. The key-based hashing algorithm successfully satisfies the three conditions required by any hashing algorithm to be secure. The algorithm uses a robust and dynamic approach towards hashing, by changing its core processes and calculating hash differently for every different combination of key and plaintext.

The first section of the paper introduces, to the reader, the concept of the paper and the requirement of a new hashing algorithm. The second section covers related literature work. The hash function is clearly explained in the third section, and its security analysis is done in the fourth section. The performance analysis of the hashing algorithm is done in the fifth section. The contributions and the applications of the hash function are covered in the sixth section. And finally, the seventh section sheds some light on the advantages of the hash function and concludes the paper.

2) Related Literature

2.1) Literature on Attacking Commonly used Key Based Hashing Methods

In their paper, Bellare, Cannetti and Krawczyk [10] discussed on keying hash functions for message authentication and also discussed several attacks on key based hashing methods. They discussed Birthday Attacks, Collision Attacks, Extension Attacks and Divide and Conquer Approach, in which they proved Nested Message Authentication Code (NMAC) to be resilient of such attacks, than key-less hashing algorithms such as MD4 and MD5.

Contini and Yin [9], in their work made staggering progress as they were able to retrieve partial key in their forgery and partial-key attacks on HMAC and NMAC. The researchers attacked the inner function in the NMAC and focused on the 34 rounds inner function in the HMAC using reduced-SHA1 to crack roughly one bit of key in every attack using collision based attacks. As predicted earlier, one of these attacks was based on pseudo-collisions of MD5, which led to pseudo-cracking of NMAC-MD5

Fouque, Leurent and Nguyen et al. [11]in their attempt to improve what Contini and Yin had acheivied, were successful in recovering the full key in HMAC-MD4 and NMAC-MD4. Their attack retrieved the Initialisation Vector, which in further helped them to recover the two keys in case of NMAC. The attack on HMAC was similar to that of NMAC with some simple modifications.

2.2) Literature on Novel Proposal of Hashing Algorithms

Noura, Salman, Chehab and Couturier [12] had proposed a novel key based hashing algorithm which was based on RC4 Stream Cipher. Even though RC4 is considered to be a weak encryption algorithm, the researchers used it as a compression function and a function to generate a dynamic independent key. They had used two features of the RC4 Cipher namely, Key Setup Algorithm and Pseudo-Random Number Generator Algorithm, as basic steps for thei proposed hash function.

Bhullar, Kumar, Pawar and Anjali et al. [13] proposed a hashing technique which was focused on removal of collisions. The use of Prime Number calculations and removal of clustering, so as to not allow data to agglomerate together at one single point, helped the target to be achieved. The main idea of derivation was from the fact that, searching algorithms cannot be calculated in O(1) time because of lack of insertion and mapping algorithms, which leads to collisions.

Chenaghlu, Jamali and Kasmakhi et al. [14] proposed a novel keyed parallel hashing system based on chaos maps. They used one dimensional and multi dimensional complex chaos maps to create a faster hashing algorithm which can provide the required results in time. Its security analysis showed its resilience to attacks such as Collision Attacks, Meet-in-the-Middle attack and Key sensitivity attacks.

2.3) Literature on Applications of Key Based Hashing Methods

Ramadhani and Ramadhani et al. [15] used One Time Pad (OTP) and Message Authentication Code (MAC) to secure the connections and communications in Whatsapp. In the OTP method, they generated a key, encrypted message, transferred the key and decrypted the message, securing communication and repeating process everytime a new message is sent. With the MAC, specifically HMAC, they used the key based hashing and OTP using same key. The new hashtext is sent, decrypted and checked against already present hashtext, hence verifying the authenticity of message.

Raihi, Bellare, Hoornaert, Naccache and Ranen et al. [16] created a One Time Password algorithm using HMAC hashing algorithm. Their algorithm was based on a hashtext value created by HMAC-SHA1 algorithm which is padded with a generated 4-byte string. Their security analysis showed that attacks on the OTP Algorithm were not possible and the best case attack was Brute Forcing Attack

Lizama-Perez [17] in their work, designed a new digital signature authentication algorithm, which was based on HMAC and called it a Singlechain algorithm. It verified the origin and the destination of a block of text and hence showed promise in replacing the Public Key Infrastructure's Digital Signature verification algorithm.

Harba [18] in their research developed a new Data Encryption Standard using pre-existing encryption methods such as symmetric AES, asymmetric RSA and HMAC with a key for secure file transfer. The AES was used to encrypt the files, RSA to encrypt the AES password and HMAC to encrypt the communication line between the server and the client.

3) How the Hash Works?

The key-based hashing algorithm is based on a generated plaintext message from the given key and plaintext inputs, called the Useful Message (UM). The key input of the algorithm is a 32 or 40 bit string which is taken as a user input alongwith the plaintext. The plaintext can be anything, and of arbitrary length. The UM is divided into parts of 64 bits each and each part is processed by the hashing algorithm and the corresponding result is stored in a variable, which is chosen, based on the cardinal of the part processed as shown in the figure below. A confusion function is applied after every 8 rounds of part-processing on the variables and the function chosen to confuse is dependent on the last bit of the last part-processed and the cardinal of the last part processed.

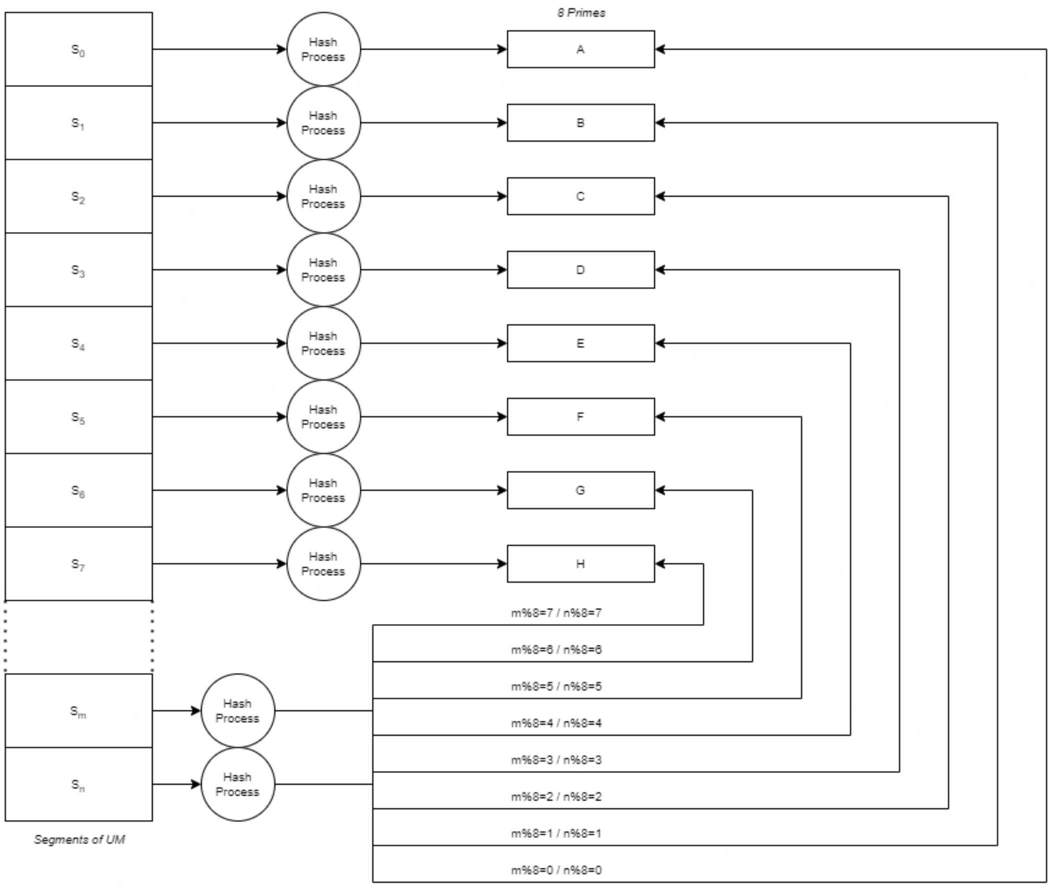

Fig1 How the algorithm works upon segments of UM

3.1) Generating the UM

The UM is the message, on which the hashing algorithm works upon. It is calculated from the plaintext and the key given to algorithm as input. The length of the UM is divisible by 64 and is longer than the length of the plaintext as well as key. The calculation of the length of UM, len_calc, is shown in Algorithm 1

Algorithm1:

```
1.  procedure len_calc(key,plaintext)
2.  start deckey <- decimal(key)
3.  l <- len(plaintext)
4.  if deckey % l == 0 and deckey>l and deckey % 64 == 0:
5.  start if num <- deckey
6.  end if
7.  else:
8.  start else while deckey % l !=0 or deckey <= l or deckey % 64 !=0 :
9.  start while deckey <- deckey+1
10. end while
11. num <- deckey
12. end else
13. end
```

Here num is the variable in which we store the length of the UM.

To calculate the UM, the randomizer bits of '101' are first padded to the plaintext. Then a collection of 64 bits, called the Appension Value (AV) is padded to the resulting plaintext. The AV is the binary form of remainder of the decimal form of plaintext and key. Then '0' bits are added to the resultant plaintext until the length of the resulting message equals to the length calculated in Algorithm1. The resultant message is called as UM and the hashing function works upon it.

3.2) Calculating 8 Primes

The resultant UM is next divided into segments of 64 bits each. Each of these segments are XORed with a variable as shown in Algorithm 3. These variables are first initialised with prime values and there are total 8 prime variables initialised. The Prime variables are initialised, using function, Primes as shown in Algorithm2

Algorithm2:

```
1.  procedure Primes(key)
2.  start deckey <- decimal(key)
3.  if deckey % 2 ==0:
4.  start if deckey <- int(pow(deckey,1/8))
5.  end if
6.  else:
7.  start else deckey <- int(pow(deckey,1/5))
8.  end else
9.  primes <- []
10. if is_prime(deckey):
11. start if primes.append(deckey)
12. end if
13. while (len(primes)!=8):
14. start while deckey <- deckey +1
15. if is_prime(deckey):
16. start if primes.append(deckey)
```

```
17. end if
18. end while
19. return primes
20. end
```

The first value of returned array primes of the function is assigned to a variable called A, second value to variable B, third value to variable C, fourth value to variable D, fifth value to variable E, sixth value to variable F, seventh value to variable G and eighth value to variable H. is_prime() is a function defined to check whether a given number as an input, is a prime number or not.

3.3) Calculating Preliminary Hashtext

Each of the segment of UM is XORed with one of the variables, initialised earlier, on basis of the cardinal of the segment. The algorithm chooses which variable to XOR as shown in Algorithm3, which depicts the function hash_calc and UM_pieces is an array which contains the 64 bit segments of the UM. The calculated XOR value is stored in the variable itself. After every 8 rounds of XORing, a confusion function is ran, which is dependent on last bit of H variable and the number of round currently being XORed.

Algorithm3:

```
1.  procedure hash_calc(UM_pieces [])
2.  start j <- 0
3.  for j in range(len(UM_pieces)):
4.  start for if j%8==0:
5.  start if A<-UM_pieces[j] ⊕ A
6.   j<-j+1
7.  end if
8.  else if j%8==1:
9.  start if B<-UM_pieces[j] ⊕ B
10. j<-j+1
11. end if
12. else if j%8==2:
13. start if C<-UM_pieces[j] ⊕ C
14. j<-j+1
15. end if
16. else if j%8==3:
17. start if D<-UM_pieces[j] ⊕ D
18. j<-j+1
19. end if
20. else if j%8==4:
21. start if E<-UM_pieces[j] ⊕ E
22. j<-j+1
23. end if
24. else if j%8==5:
25. start if F<-UM_pieces[j] ⊕ F
26. j<-j+1
27. end if
28. else if j%8==6:
29. start if G<-UM_pieces[j] ⊕ G
30. j<-j+1
31. end if
32. else if j%8==7:
33. start if H<-UM_pieces[j] ⊕ H
34. j<-j+1
35. end if
```

```
36. if j%8==0 and last_bit(H)==0:
37. start if A,B,C,D,E,F,G,H = fun0(A,B,C,D,E,F,G,H,j)
38. end if
39. else if j%8==0 and last_bit(H)==1:
40. start if A,B,C,D,E,F,G,H=fun1(A,B,C,D,E,F,G,H,j)
41. end if
42. end for
43. return A+B+C+D+E+F+G+H
44. end
```

The above algorithm returns a 512-bit output which is a concatenation of final values of all the eight variables. last_bit() is a function which determines the last bit of a given input variable. fun0 and fun1 are the two confusion functions described earlier wherein fun0 is called if the last bit of H is 0 and fun1 is called if last bit of H is 1. fun0 and fun1 process the inputs differently for different values of j, which here represents the number of segments processed. If j is divisible by 24, the process is different, if it is divisible by 16, the process is different and a different process when j is divisible by 8.

3.4) 1-Formatting

1-Formatting in the algorithm is a special process in which certain bits of the output hash are converted to '1' bit irrespective of their previous value. 1-Formatting makes it virtually impossible to crack the hash code, without having access to the plaintext itself. In 1-Formatting, all the factors of plaintext's decimal form, smaller than 512 are gathered and the bits at those particular places, are converted to '1' bit irrespective of the bit already present over there. It helps is diffusing the properties of plaintext over the resulatant hashtext. The binary text is then converted to hexadecimal hashtext and presented in form of the algorithm's final output.

4) Security Analysis

Several security checks and tests were carried out on the algorithm to ensure it is resilient to attacks such as pre-image and collision attacks. The algorithm proved to be robust and dynamic when tested under different attacks of Brute Forcing, Backtracking Attack and based on attacks proposed by other researchers.

4.1) Brute Forcing Attacks

Brute Forcing refers to trial and error method to guess the correct pre image of a given hashtext or find a collision for a given hashtext. There are various methods of Brute Forcing attacks in use today, two of which have been tried and tested upon the algorithm. The complete brute forcing attack was further divided into two sub sequent attacks: Plaintext Brute Force and Key Brute Force.

4.1.1) Dictionary Attack

Dictionary Attack refers to a Brute Force attack, which includes a predefined list of inputs, against which the algorithm to be attacked, is checked upon **Error! Reference source not found.**. To simulate a Dictionary Attack on the algorithm, a fixed key was considered and a hashtext was produced using that key and a word from English Dictionary as plaintext. Then, a brute force attack was placed where a hashtext was generated using each and every word from the dictionary and was tallied with the original hashtext to check whether it matched or not. This attack was carried out ten times over a course of a week and the results were given input to a Machine Learning Algorithm to predict the time taken by Brute Force to reveal the preimage of the hashtext. The results from Machine Learning Algorithm are depicted in the graph below:

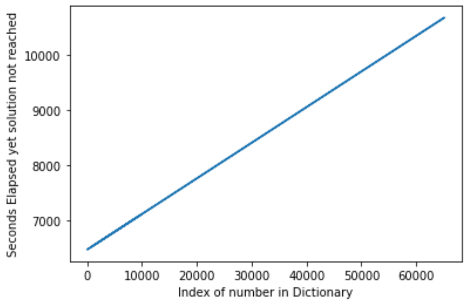

Fig 2 ML Output of Dictionary Attack

The above graph represents the inadequacy of a Dictionary Attack showing that how the attack takes more than 2 hours, even for words having lower index numbers than other. As the researchers had not spent more than 10800 seconds per experiment, the results of the algorithm are limited to the same, but still the attack was not able to produce satisfactory results.

4.1.2) Plaintext Brute Force

The Dictionary Attack was limited to the words from English Dictionary. In Plaintext Brute Force, the plaintext can be any combination of the 256 ASCII characters and of any length. Assuming from the Dictionary Attack results, a Plaintext Brute Force attack seems to be virtually impossible, but was still devised to crack the hashing algorithm. A plaintext of length 8 characters (64-bits) was created and hashed with a fixed key. To keep things simple for the attack, the plaintexts, against which the algorithm was checked upon, were limited to size of 64 bits. A random string generator generated a string of 8 characters and hashed it using the fixed key, tallied the hashtext against the original output and tried to crack the hashing algorithm. The output of the time elapsed in bringing the output, yet the result not being satisfactory has been depicted in the below graph:

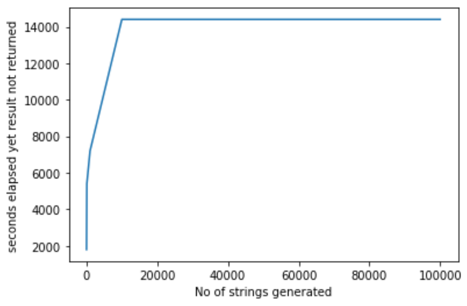

Fig 3 Output of Plaintext BruteForce

4.1.3) Key Brute Force

The results of Plaintext Brute Force and Dictionary Attack prove the hashing algorithm to be resilient of Brute Force Attacks, but the key can be brute forced as well. Hence a key-brute force attack was also carried out. A list of all possible keys was created and a key was chosen to hash a fixed plaintext. The attack involved hashing the fixed plaintext with all the possible keys and checking the resultant hashtext with original output hashtext. The key bruteforce attack can be deemed infeasible because of such huge number of keys possible. As calculated, the algorithm takes O($k^{3/2}$) to generate a hashtext for a single key where k is the decimal value of key. k assumes the lowest decimal value of 8788065 and highest decimal value of 34089189246, hence rendering the key brute force infeasible. The results of key brute force are depicted in the graph below:

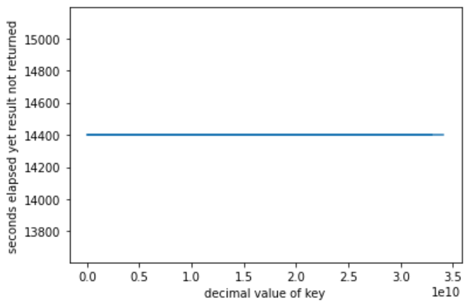

Fig 4 Results of Key Brute Force Attack

4.2) Backtracking Attack

The backtracking attack is based on retrieving the plaintext from the hashtext with or without the knowledge of the key. Backtracking attacks have usually failed in cracking hash codes but have certainly helped many researchers in cracking encryption functions. The processes which can be carried out and are not limited to, in order to break the proposed hashing algorithm:

4.2.1) Breaking 1-Formatting

Breaking 1-Formatting in hashtext is a tricky job, even with the presence of key. It asks for a time complexity of O(2^n) where n is the number of factors of plaintext. We can create a powerset of all the factors and start trying the cracking process by converting the 1s at those positions to 0. But the real obstacle in cracking is the uncertainty whether the bit being changed was definitely 0 or not before applying 1-Formatting. Considering n to be a small number, such as 2 or 3 (minimum, n needs to be 2 in case of a prime plaintext), 1-Formatting problem can be cracked into several different hashtexts, each of them needed to be solved individually.

4.2.2) Backtracking UM Cycle

The bigger problem lies in solving each and every hashtext individually, in case of 3 factors of key, we will have 8 hashtexts possible. The hashtext will be further broken into 8 equal parts, which are the last values of A to H. But this diverges in 10 different directions, which of the prime was last modified and if it were H, was fun1 used or fun0 used to further modify A to H. To backtrack, we do have one element, i.e., the values of A to H, but we don't have another element, which is 64-bit pieces of the UM. One can randomly make an educated guess of the number of rounds the UM would have to go through to give the final hashtext, but they cannot guess the exact position where the real message ends and appended message start as they would have no knowledge of AV or the length of the message. Hence, backtracking UM Cycle becomes even more complex procedure and cannot be cracked easily, hence providing the security to our plaintext and information.

4.3) Attacks by other Researchers:

Our hashing algorithm was tested against attacks carried out by other researchers as well (discussed in Section 2.1). The results were found to be promising and the hashing algorithm was secure against those attacks. There were many attacks discussed in [10] such as Birthday Attacks, Collision Attacks and Extension Attacks. Our hashing algorithm is resilient to Birthday Attacks as algorithms with longer output have more resiliency to Birthday Attacks than those with smaller outputs [10]. The authors of [10] proved HMAC with 128-bit output to be resilient to Birthday Attacks, while our hashing algorithm gives an output of 512-bits, which is four times longer than HMAC output and hence more secure as well. Due to its length the collision chances also go down to 2^{-512} hence making chances of collision negligible. Their proposed Extension attack and the Divide and Conquer Approach fail on the hashing algorithm as the avalanche effect works perfectly on the output hashtext as rarely more than 2 characters occupy the same place in hashtext with a minor change in plaintext.

The attacks carried out in [9] and [11] were highly successful because the HMAC and NMAC are dependent on pre-existing insecure hashing algorithms such as MD5 and MD4. The attackers were able to retrieve the key partially in [9] and fully for certain underlying hashing algorithms in [11]. The attack in [9] were successful because of attacking fixed number of 34 rounds in MD5 algorithm and the attack in [11] were successful because of a fixed IV used in NMAC and HMAC. This attacks nullify on the proposed hashing algorithm as there is no particular fixed number of rounds in the process (the process might not have even 34 rounds to calculate hashtext or it can even have a huge number of rounds, such that 34 becomes negligibly small). The second attack fails as the algortihm doesn't use a fixed IV. The appended message and the length of the message in the algorithm depends on both the Message and the Key (ie. AV) and hence are secure from attacks discussed.

5) Performance Analysis

The proposed hashing algorithm does provide better security standards than existing key-based hashing algorithms, but it also needs to be computationally sound and lightweight to be applied in daily uses. The steps carried out in calculating the hashtext are as follows:

- Converting the key and plaintext to corresponding binary values
- Calculating length of UM

- Generating UM
- Calculating the 8 Primes
- Dividing UM into 64 bit segments
- Calculating primitive hashtext
- 1-Formatting

Hence the Computational Delay (CD) can be calculated as:

$$CD = T_{len\_calc()} + T_{Primes()} + T_{hash\_calc()} + T_{1\text{-}Formatting} \qquad (1)$$

Where T_n refers to the Time taken to execute n function. The largest time amongst the given functions is consumed by the Primes() function, which is $O(k^{3/2})$ where k is the decimal value of key and the other values are usually negligible infront of it.

6) Applications of Hashing Function

A hashing function can be of utilitarian purpose, if and only if it finds its uses in various fields of science and real-time applications**Error! Reference source not found.**. Hashing functions such as SHA-256 and SHA-512 became useful only after they found their application in Blockchain Technology. Hashing Algorithm such as MD5 became purposeful only after it found its application in file hashing and checksums. So, for the proposed hashing algorithm to be useful, it must have some applications in the field. Some prospective applications for the algorithm asre discussed below.

6.1) Password Storage and Authentication

One of the major use-cases can be used by websites which store passwords and usernames of their users to authenticate them. Hashing technology such as SHA-512 and MD-5 had been used widely for password verification in earlier times. But using a key-based hashing algorithm gives the authenticator an upper hand over the attackers. Without the knowledge of key, the attackers cannot replicate the hashtext with some random plaintext and try to attack the hashing algorithm based on collision.

The algorithm gives an advantage to the authenticators as:

- A 512-bit hashtext gives virtually the same security as SHA-512 (which also provides a 512-bit output), but an advantage over the same with the anonymity of the key.
- The value of the key is only known to the authenticator; hence the authenticator becomes the moderator of the algorithm.
- Even if the key and hashtext are compromised to the attacker, it becomes virtually impossible for the attacker to retrieve the plaintext as discussed earlier.

The algorithm drastically improves the password storage and authentication mechanism as it is fast for smaller keys, requires less storage space than other key-based hashing system and provides security tantamount to one of the most secure hash functions in use today, but with an added advantage of anonymous key.

6.2) Web3.0 and Decentralised Storage

The advent of Web3.0 has been welcomed in the world of technology with a lot of applause. The founding fathers of Web3.0 decided to incorporate security of data as one of the major key points in their invention. Hence, the notation of Decentralised Storage came into existence. The files to be stored are usually broken down into several small pieces of equal size with each piece being located separately and independently of one another. These pieces can be tracked down only by the Web3.0 and the service provider of storage. The proposed hashing algorithm can play a pivotal role in the development of Decentralised Storage as a secure storage service. Each of the small piece of file can be independently hashed into a hashtext and stored in the storage, which ensures avoidance of illicit access to sensitive data. The hashing algorithm gives the following advantages in Decentralised Storage:

- Independent hashing of each piece, independent of one another and making it impossible for attacker to even retrieve a single piece of information
- Easy authentication by storage service provider due to anonymous key value
- Easy access to original owner of the file without compromising security
- Easy sharing of data by owner to authenticated users by sharing the unique key per stored item.

6.3) Miscellaneous Applications

The algorithm can find its footing in many miscellaneous applications in the world of technology, whether it is in the field of Internet of Things or Artificial Intelligence. The major reason behind it is, requirements in security increasing every day, due to advancements in cyberattacks. A robust and sturdy hashing algorithm is of utmost importance right now which cannot be cracked under any circumstanes and the proposed hashing algorithm satisfies the requirements. It can be applied in following fields of technology:

- *Internet Of Things*: To ensure secure access and command of internet-connected things by rightful users
- *Artificial Intelligence*: To ensure safe and authorised access to Smart agents working on certain projects
- *High Threat and Highly Confidential Projects*: Projects sensitive to military and government prospectus need to be kept safe and secure from the hands of people with malicious intent. The algorithm ensures a high-level protection and security for same.
- *Image and Video Hashing*: Hashing can be applied to a higher domain than only text. Image files, video files and even audio files can be hashed as it can also be items of interest that common public would like to share and store (as mentioned in Section 6.2)

7) Conclusion

The proposed hashing algorithm is a secure and attack-resilient key-based hashing algorithm which can find its footing in many security applications today. Its efficiency lies in the small amount of storage it requires and the time it takes for traditional approaches to crack the algorithm for either pre image attack or collision attacks. It is strikingly different from key-based hashing algorithms such as HMAC, NMAC or GMAC as it does not rely upon previously existing hashing algorithms and improve their security. The methodology of its hashing has been divided into different, dynamic steps, with the output of each step being input to other steps and all the outputs collectively working together to bring the final hashtext.

The algorithm can play a pivotal role in Web3.0 and Decentralisation as well as in secure storage of files and passwords, providing means of authentication as well. It can also help in boosting the IoT boom as well as AI boom in the upcoming years, due to the high level of security it provides. It is not only resilient to brute forcing or backtracking attacks, but also resilient to attacks proposed by other researchers on basic key-based hashing algorithms, providing the algorithm an edge over them.

Due to high amount of prospective applications in the future, along with a high level of security and other salient features provided by the algorithm, it can be concluded that the algorithm can be replaced as a new, alternate key-based hashing algorithm in everyday applications.

8) References

[1] Turner, J. M. (2008). The keyed-hash message authentication code (hmac). *Federal Information Processing Standards Publication*, *198*(1), 1-13.

[2] Krawczyk, H., Bellare, M., & Canetti, R. (1997). *HMAC: Keyed-hashing for message authentication* (No. rfc2104).

[3] Song, F., & Yun, A. (2017, August). Quantum security of NMAC and related constructions. In *Annual International Cryptology Conference* (pp. 283-309). Springer, Cham.Rachmawati D, Tarigan JT, Ginting ABC: A

comparative study of Message Digest 5(MD5) and SHA256 algorithm. 2nd International Conference on Computing and Applied Informatics 2017 28-30 November 2017. (2017)

[4] McGrew, D., & Viega, J. (2006). *The use of galois message authentication code (GMAC) in ipsec ESP and AH* (No. rfc4543).Pieprzyk, J., & Sadeghiyan, B. (Eds.): *Design of hashing algorithms*. Berlin, Heidelberg: Springer Berlin Heidelberg. (1993)

[5] Kim, J., Biryukov, A., Preneel, B., & Hong, S. (2006, September). On the security of HMAC and NMAC based on HAVAL, MD4, MD5, SHA-0 and SHA-1. In *International Conference on Security and Cryptography for Networks* (pp. 242-256). Springer, Berlin, Heidelberg.Turner,

[6] Tsai, J. L. (2008). Efficient multi-server authentication scheme based on one-way hash function without verification table. *Computers & Security*, *27*(3-4), 115-121.

[7] Preneel, B. (1993). *Analysis and design of cryptographic hash functions* (Doctoral dissertation, Katholieke Universiteit te Leuven).

[8] Guo, J., Peyrin, T., Sasaki, Y., & Wang, L. (2014, August). Updates on generic attacks against HMAC and NMAC. In *Annual Cryptology Conference* (pp. 131-148). Springer, Berlin, Heidelberg.

[9] Contini, S., & Yin, Y. L. : Forgery and partial key-recovery attacks on HMAC and NMAC using hash collisions. In *International Conference on the Theory and Application of Cryptology and Information Security*, Springer, Berlin, Heidelberg. (2006) 37-53

[10] Bellare, M., Canetti, R., & Krawczyk, H. : Keying hash functions for message authentication. In *Annual international cryptology conference*, Springer, Berlin, Heidelberg. (1996) 1-15

[11] Fouque, PA., Leurent, G., Nguyen, P.Q. :Full Key-Recovery Attacks on HMAC/NMAC-MD4 and NMAC-MD5. In: Menezes, A. (eds) Advances in Cryptology - CRYPTO 2007. CRYPTO 2007. Lecture Notes in Computer Science, vol 4622. Springer, Berlin, Heidelberg. (2007)

[12] Hassan Noura, Ola Salman, Ali Chehab, Raphael Couturier: Efficient and Secure Keyed Hash Function Scheme Based on RC4 Stream Cipher. Symposium on Computers and Communications, Jul 2020, Rennes, France. (2020)

[13] R. K. Bhullar, L. Pawar, V. Kumar and Anjali: "A novel prime numbers based hashing technique for minimizing collisions," 2016 2nd International Conference on Next Generation Computing Technologies (NGCT) (2016) 522-527

[14] Chenaghlu, M. A., Jamali, S., & Khasmakhi, N. N.: A novel keyed parallel hashing scheme based on a new chaotic system. *Chaos, Solitons & Fractals*, *87* (2016) 216-225.

[15] Ramadhani, F., Ramadhani, U., & Basit, L.: Combination of Hybrid Cryptography In One Time Pad (OTP) Algorithm And Keyed-Hash Message Authentication Code (HMAC) In Securing The Whatsapp Communication Application. *Journal of Computer Science, Information Technology and Telecommunication Engineering*, *1*(1), (2020) 31-36.

[16] M'Raihi, D., Bellare, M., Hoornaert, F., Naccache, D., & Ranen, O. :*Hotp: An hmac-based one-time password algorithm. RFC 4226* (2005)

[17] Lizama-Pérez, L. A. : Digital signatures over HMAC entangled chains. *Engineering Science and Technology, an International Journal*, 101076. (2021)

[18] Harba, E. S. I. : Secure data encryption through a combination of AES, RSA and HMAC. *Engineering, Technology & Applied Science Research*, *7*(4), (2017) 1781-1785.

[19] Bošnjak, L., Sreš, J., & Brumen, B. (2018, May). Brute-force and dictionary attack on hashed real-world passwords. In *2018 41st international convention on information and communication technology, electronics and microelectronics (mipro)* (pp. 1161-1166). IEEE.

[20] Bakhtiari, S., Safavi-Naini, R., & Pieprzyk, J. (1995, July). Keyed hash functions. In *International Conference on Cryptography: Policy and Algorithms* (pp. 201-214). Springer, Berlin, Heidelberg.