INTERNATIONAL JOURNAL OF ADVANCED STUDIES
IN COMPUTER SCIENCE AND ENGINEERING
IJASCSE VOLUME 6 ISSUE 2, 2017

02/28/2017

Variable Length Hash Algorithm using Random Number

Devendra Kumar Sahu
Research Scholar,
Samrat Ashok Technological Institute,
Vidisha (M.P.)

Ramratan Ahirwal
Assistant Professor,
Samrat Ashok Technological Institute,
Vidisha (M.P.)

***Abstract*- Integrity is very important during receiving of stored and transmitted data. Standard, SHA-1 algorithm used to ensure the integrity using fixed length digest but a mathematical collision found on SHA-1. Later, RC6_Hash algorithm et al [1] provided a technique in which it generates a variable size digest using a symmetric block cipher. It is time efficient but not secure enough as it has very low avalanche effect. In this paper a new hash algorithm is proposed with the same objective in which it generates variable size digest. Proposed algorithm uses a symmetric block cipher which is based on a random number which makes this algorithm time efficient and provides strength against any attack. Block cipher is used as a hash function to generate digest. Proposed algorithm uses digest of first block as a key to second block and so on. This increases the avalanche effect. The presented algorithm have time efficiency as well as enough secure to prove integrity over any received data.**

Keywords- Computer Security, Data Integrity, Hash Algorithm, Message Digest

I. INTRODUCTION

Today, as we are growing towards modern age, our requirement also changes very rapidly. We all are very much dependent on digital world now days, for storing data, communication, entertainment, business services etc. All of these are the necessary but with security such that no one other can access or alter the stuff for which they are not authorized. Whenever a discussion of security comes, three terms comes in mind: confidentiality, integrity and authentication. Confidentiality ensures that transmitted data or stored data cannot be readable by any other un-authorized person. Integrity ensures that there is no change in data during transmission or storage by intruder or may be by noise. Authentication on the other end ensures that no unauthorized person transmit or gain confidential data.

A. INTEGRITY

Integrity means ensure that the data we received, is the same, that was transmitted or stored by source end, if there is a change, no matter whether it is minor changes or major changes, either by intruder or by noise, it will be detected by the algorithm. Integrity is ensuring by using Hash Algorithm.

B. HASH ALGORITHM

Hash algorithms are designed in such a way that it generates a digest (similar to signature) of any input data which need to be transmit or stored.
Message digest is an encrypted text containing a string of characters generated by a one-way hashing algorithm. Message digest is generated from message itself by performing some operations over it. This digest is totally different for two different data having minor changes.

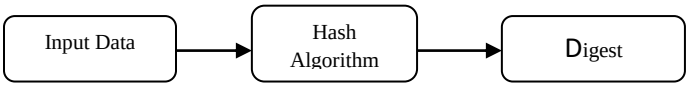


Figure 1 Process of generating digest at source end

Then after generating the digest, this digest and the original data transmit or stored together. Figure 1 and Figure 2 shows how hash algorithm ensures the integrity over data.

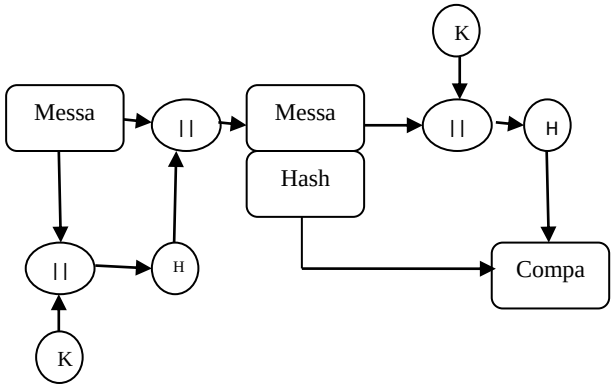


Figure 2 Basic Hash Algorithm at sender and receiver

At receiving end, we received both original data and its digest as shown in Figure 2, received data again passed to the same hash algorithm which generates a digest again. If this newly generated digest is same against received digest, it means there is no change in the data but if digest are different it means some changes had made in data either by noise or by intruder.

The hash algorithm must have the following four properties:

i. It should be ease to generate digest for any given input string.
ii. It should be impossible to generate message from digest. It should be uni-directional.
iii. Hash should be designed in such a way that any changes in message should change the digest completely.
iv. It should be impossible to find two different messages having same digest.

II. LITERATURE SURVEY

Number of algorithms has been designed to ensure the integrity in received data. SHA family et al [5] is the most popular for ensuring integrity. SHA family consist three different structures SHA-0, SHA-1 and SHA-2. But an attack in year 2008, found a collision on SHA-0 and in year 2011, an attack by Marc Stevens produces a collision on SHA-1.

No successful attacks have been found on SHA-2 till now, but still SHA-2 algorithms did not gain much popularity because of its inefficiency against time.

After these collisions found on SHA family, a gap is created in data integrity. Many algorithms are proposed to fill that gap, but none of them succeed to design it. In order to improve integrity they compromise with the time efficiency and vice versa.

Vikas Tiwari et al [3], identified the integrity problem, and proposed their own algorithm which actually a combination of SHA-1 and MD-5. Because the existing SHA-1 was proven breakable so in order to improve the security they merged SHA-1 with MD5. This concept improves the robustness of the algorithm but at the same time it decreases the efficiency in time which is now the sum of time to run SHA-1 and MD5.

Kirti Aggarwal et al [2], proposed an algorithm which is the modification of standard SHA-1. They named there algorithm SHA-192 which generates a digest of 192 bits. In order to increase the security, they increase one chaining variable in SHA-1 of 32 bit.

Kirti Aggarwal et al [1] proposed there research in which they have designed an algorithm that can generate variable size digest using RC6 algorithm. They named there algorithm RC6 Hash. The key feature of this algorithm is variety in its digest. Before this all algorithms generate fixed sized digest but RC6 Hash moves one step forward and generate a variable size digest.

This paper has implemented standard SHA -1 and RC6 Hash and found that the algorithm has high time efficiency than SHA-1 but not secure. Changing in a text, generate a same digest which make it futile.

The authors of this paper have proposed their own algorithm in order to fit a proper solution of above discussed problem. The proposed algorithm have been prepared a long research on existing algorithms with their merits and demerits and proved with their implemented results that the proposed work is one of the best solutions.

III. METHODOLOGY OF PROPOSED WORK

Challenge for designing a hash algorithm is that it has low time for digest generation and high avalanche effect. The concept of avalanche effect refers to change in number of bits in digest corresponding to change in message.

Proposed algorithm is designed in such a way that it generates a variable size digest by using symmetric block cipher. This algorithm works on random number which makes this algorithm time efficient and provides strength against any attack.

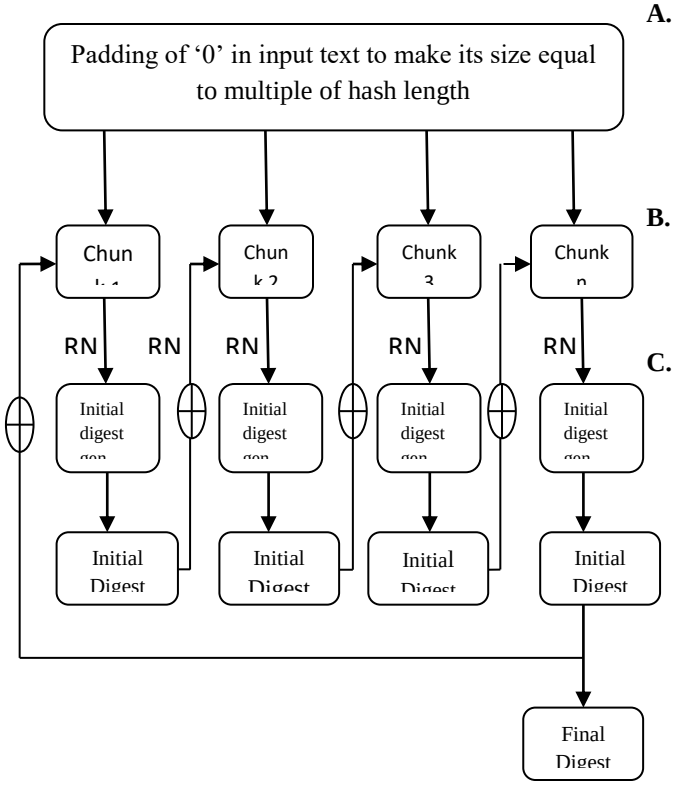


Figure 3 Generation of digest by proposed algorithm

In proposed algorithm symmetric block cipher algorithm is used as a hash function et al [12]. Here, block cipher is used in Cipher Block Chaining (CBC) mode in which the output digest of previous block is input to the next block.

In our proposed hash algorithm we first add the padding bits at the end of message to make the message size equal to the length multiple of 256 bits, where 256 bits are the length of hash/digest value. After that we divide the entire message into n number of chunks of size equal to hash value.

Now, we get the initial digest for first chunk. To get the initial digest of first chunk we use a random number generated itself by chunks and perform initial digest generation over it. Later on we perform XOR operation with initial digest of each chunk to the next input chunk (before applying initial digest generation on next chunk). This whole operation is repeated two times to find the final digest in our proposed method. As we know that if the block algorithm is secure, then the one way hash function will also be secure. In this way our algorithm provides better integrity.

A. Padding: Padding is added to make message length equal to multiple of hash length, thus we add number of 0's as padding bits at the end of input text. If input text is already in multiple of hash length then addition of padding bits are not required. For example, if message is of 1000 bits and hash value is 256 then 24 zeros are added as padding bits to make it 1024 bits.

B. Random number generation using chunk: In proposed algorithm random number (RN) is required to generate the initial digest. This random number is generated itself by chunk by counting number of '1' in input chunk. For example, chunk= 10010101, then RN= 4.

C. Initial Digest Generation: To find the initial digest of each chunk, we perform XOR operation based on random number. If the counted value of random number is RN, then the first positioned bit from left is XORed with the RN positioned bit from left. Similarly 2^{nd}, 3^{rd}……. nth bits from left are XORed with RN+1, RN+2,…… RN+n, bits positioned at chunk with circular move. For example, if chunk=100010 then RN=2, now XOR each bit of chunk with the next 2^{nd} bit, i.e. $1\oplus0$, $0\oplus0$, $0\oplus1$, $0\oplus0$, $1\oplus1$, $0\oplus0$. The result is 101000.

Then we divide XORed chunk into left and right halves. Now swap left and right halves with each other. After that we perform the left circular rotation by half of the random number bits counted for the chunks on left half of the chunk. Now, the result of left circular rotation is XORed with right half and result becomes the right half of chunk. Then concatenate both halves.

After generating initial digest for first chunk, the initial digest of first chunk is XORed with input text of second chunk. Then we perform initial digest generation method for the XORed chunk. Similarly, for each chunk, input texts is XORed with the initial digest of previous chunk and perform initial digest generation method to get the final digest of first round.

for each chunk: i=2

{

$Chunk_i = Chunk_i \oplus Initial\ Digest_{i-1}$;

Initial Digest Generation ($Chunk_i$);

}

Now, divide the final result of first round into two parts and swap them. Than we perform XOR the final result with the first chunk of an input text and repeat the whole process for second round so that last chunk also shuffled well. The final result of second round is the desired message digest of our proposed algorithm.

The final algorithm comes out at each chunk for generating hash of a given text is:

```
for (int i = 0; i < txt.Length; i =
i + x)
{
   hash = xor(hash,
con_text_to_Bin(txt.Substring(i,
x)));
   RN = GetRandom(hash);
   hash = R_Ran_xorbits(hash, RN);
   hashR = hash.Substring(0, HL /
2);
   hashL = hash.Substring(HL / 2);
   hashL = leftrotate(hashL, RN /
2);
```

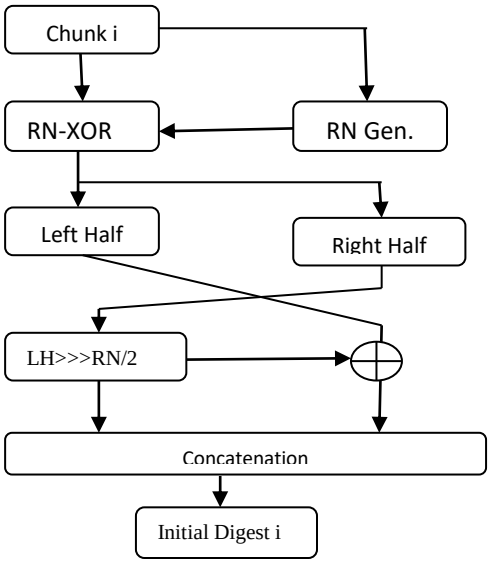


Figure 4 Generation of digest for each chunk

IV. EXPERIMENTAL RESULTS

The implementation result of proposed algorithm is discussed and compared it with standard SHA -1 algorithm et al [5] and RC-6 Hash algorithm et al [1] in this section. Implementation of all three algorithms is done in DOT NET Framework. The testing was performed on a system having Intel Pentium Dual Core E2200 2.20 Ghz, 1 GB of RAM and Window XP Service pack 2 configurations.

A. STRENGTH ANALYSIS

It is important to calculate the internal strength of an algorithm. To calculate the internal strength of proposed algorithm, SHA-1 and RC6 Hash, avalanche effect of all the three algorithms is calculated. The concept of avalanche effect says that on changing a single bit in input text, change the digest fifty percent, but this is an idle condition. An algorithm which is much closer to avalanche effect condition assumes to be more secure than other.

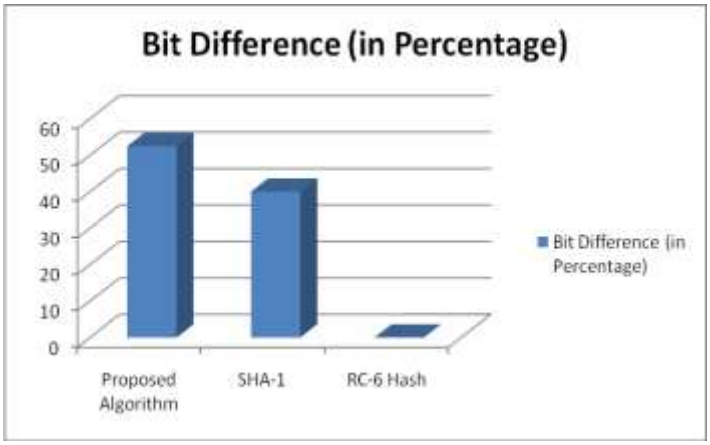
Table 1 shows the avalanche effect on a two different text having a single character change.
Text 1: “the quick brown fox jumps over lazy dog”
Text 2: “ehe quick brown fox jumps over lazy dog”
Avalanche effect =Number of different bits ($Digest_{Text\ 1}$, $Digest_{Text2}$)

Steps to generate a hash using proposed algorithm are as follows:

1. First take a hash length as an input from a user and make a text in multiple of hash length by padding ‘0’ at the end of text.
2. Now divide the text into number of chunks where each chunk is of length equal to hash length.
3. Next for each chunk do the following:
   a. Convert the text into binary format
   b. Generate a random number (RN) by simply adding the entire one’s in a chunk.
   c. Now, perform XOR all the bits of a chunk by its next bits at position R.
   d. Next divide the chunk into two equal half and swap first half with the next half.
   e. Now, perform left circular rotation on first half by R/2 bits.
   f. Perform XOR of first half with second half and result becomes the second half of chunk.
   g. Now, concatenate both half
4. Repeat Step3 for each chunk and after completion of each chunk the result of previous chunk is XOR with input text of next chunk.
5. After completion of all chunks, swap the first half with second half of final result.
6. XOR again the final result with the first chunk of an input text and repeat Step 3 for all input text chunks.
7. The final result comes out after applying Step 6 is hash value of give input text.

TABLE 1 AVALANCHE EFFECT OF PROPOSED ALGORITHM, SHA-1, RC-6 HASH ALGORITHMS

Algorithm	Avalanche Effect	
	Bit Difference (in Percentage)	Difference from Idle condition
Proposed Algorithm	52.5	+2.5%
SHA-1	40	-10%
RC-6 Hash	0	-50 %



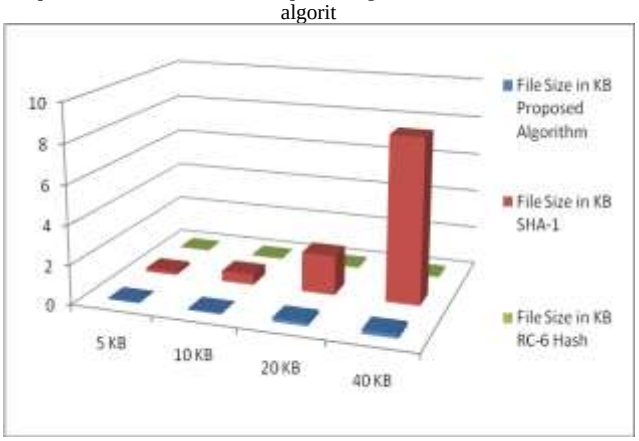
Graph 1: Avalanche effect of Proposed Algorithm, SHA-1, RC-6 Hash algorit

Now, from Table 1 and its graphical representation Graph 1, it is clear that proposed algorithm is much closer to avalanche effect concept compare to other SHA-1 and RC-6 Hash.

Timing Analysis: It is also important for an algorithm to be time efficient. As modern technology changes, requirement of using Ad-Hoc network or wireless devices are increases very rapidly. Time efficient algorithms are suitable for such devices. Not only for this but also time are efficient algorithms required for real time communication. Table 2 shows the experimental results and compared the time required by proposed algorithm, SHA-1 and RC-6 Hash for a same file.

TABLE 2 Timing Comparison between Proposed Algorithm, SHA-1 and RC-6 Hash algorithms

File Size in KB	Algorithms (Time in Seconds)		
	Proposed Algorithm	SHA-1	RC-6 Hash
5 KB	0.031	0.187	0.015
10 KB	0.062	0.546	0.046
20 KB	0.187	2.028	0.064
40 KB	0.202	8.361	0.072



Graph 2 Timing Comparison between Proposed Algorithm, SHA-1 and RC-6 Hash algorithms

It is clearly seen from the above table that proposed algorithm is highly time efficient than SHA-1 and RC-6 Hash is time efficient than proposed algorithm. Graphical representation of TABLE 2 is shown in Graph 2.

V. CONCLUSION

In this paper a new hash algorithm for ensuring integrity in received data is proposed. The proposed algorithm is compared with Standard SHA-1 algorithm which generates fixed size hash and RC6 hash which generate variable length hash using block cipher. It is found that proposed algorithm is highly secure as compared to SHA-1 and RC-6 Hash. RC-6 Hash is time efficient but it does not provide integrity as it produces no change in digest

while changing in input text. Implementation results clearly show that proposed algorithms completely fill the gap that was created due to SHA-1 collision.

Future Scope: Algorithm can further optimize by implementing it on hardware and can be merge with other algorithms to gain benefits like authenticity or confidentiality.

REFERENCES:

[1] Kirti Aggarwal, Dr. Harsh K. Verma, "Hash_RC6 - Variable Length Hash Algorithm using RC6", 2015 International Conference on Advances in Computer Engineering and Applications (ICACEA) IMS Engineering College, Ghaziabad, India, IEEE
[2] Kirti Aggarwal, Jaspal Kaur Saini, Dr. Harsh K. Verma "Performance Evaluation of RC6, Blowfish, DES, IDEA, CAST-128 Block Ciphers", International Journal of Computer Applications, 2013.
[3] Vikas Tyagi, Shrinivas Singh, Volume 3, No. 4, April 2012 Journal of Global Research in Computer Science "Enhancement of RC6 (RC6_EN) Block Cipher Algorithm and Comparison with RC5 & RC6.
[4] "Hash Functions". Cse.yorku.ca. September 22, 2003. Retrieved November 1, 2012. "The djb2 algorithm (k=33) was first reported by dan bernstein many years ago in comp.lang.c.
[5] Behrouz A. Forouzan, Debdeep Mukhopadhyay, "Cryptography and Network Security", Tata McGraw Hill Education Private Limited, NEW DELHI, 2010.
[6] Thulasmani, L. Madheswaran, M. "Security and Robustness Enhancement of Existing Hash Algorithm"
[7] Published in IEEE International Conference on Signal Processing Systems 15-17 May 2009 Page(s):253 - 257 Print ISBN:978-0-7695-3654-5
[8] What are RC5 and RC6","rsa.com". Available at: http://www.rsa.com/rsalabs/node.asp?id=2251.The Collision Rate Tests of Two Known Message Digest Algorithms 2009.
[9] H. Mirvaziri, Kasmiran Jumari, *Mahamod Ismail, Z. Mohd Hanapi, "*A new Hash Function Based on Combination of Existing Digest Algorithms*",* The 5th Student Conference on Research and Development –SCOReD 2007 11-12 December 2007, Malaysia, IEEE
[10] W. Stallings, "Cryptography and Network Security: Principles and Practice", Prentice-Hall, New Jersey,1999.B. Schneier, "Applied Cryptography", John Wiley & Sons Inc., 1999.
[11] R.L. pavan, M.J.B. Robshaw, R.Sidney, and Y.L. Yin. The RC6 Block Cipher. v1.1, August 1998.
[12] Descriptions of SHA-256, SHA-384, and SHA-512 www.iwar.org.uk/comsec/resources/cipher/sha256-384-512.pdf.
[13] "Hash Function and Block Cipher", http://burtleburtle.net/bob/hash/index.html.
"Cryptographic hash function", http://en.wikipedia.org/wiki/Cryptographic_hash_function