Available online at www.scientiaresearchlibrary.com

Scientia Research Library

ISSN 2348-0424
USA CODEN: JETRB4

Journal of Engineering And Technology Research, 2019, 7 (1):1-10
(*http://www.scientiaresearchlibrary.com/arhcive.php*)

A COMPARATIVE ANALYSIS OF MODERN CRYPTOGRAPHIC HASH FUNCTIONS

FERNANDO S*. VIRAY JR.
[1]*Pangasinan State University*
Corresponding Author: [1]*skyfher@gmail.com*

ABSTRACT

The need to provide a short yet computationally infeasible message digest as security reference is tantamount to secured sending of information or files in an unsecured network. Cryptographic hash functions are one-way functions that yields a 32-character length string that is used to validate and cross-check that information received is indeed unchanged when it travelled in an unsafe communication channel. In this paper, the researchers evaluated the two most common modern cryptographic hash functions, Message Digest number 5 (MD5) and Secure Hash Algorithm number 1 (SHA-1) in terms of performance and computational hardness.

Keywords : modern cryptographic hash function, Message Digest 5 (MD5), Secure Hash Algorithm (SHA-1)

INTRODUCTION

Cryptography prior to the modern age has been synonymous with encryption - the conversion of information from a readable state to garbled texts which are apparently nonsense. Its study, cryptology, is aimed heavily towards techniques for secure communication[1]. However, as cryptographers and cryptologists invented several ways to encrypt information, loopholes are observed and are most of the times affected by how the message was delivered and by what channel is used to arrive to the purported receiver of the information. Thus, a wider scope of cryptography towards secure communication which established the fact that along the channel of communication is the presence of third parties who are referred to by Ronal L. Rivest in 1990 as “adversaries.” The cryptography literature often uses Alice ("A") for the sender, Bob ("B") for the intended recipient, and Eve ("eavesdropper") for the adversary [2].

The consideration of the channel as an integral factor in a secure communication brought the rise of modern cryptography which utilizes the combination of different cryptographic techniques in encrypting information and securely delivering the encrypted message to the receiver. These techniques are called cryptosystems. It is a suite of cryptographic algorithms needed to implement a particular security services which fall within the central aspects of modern cryptography - data confidentiality, data integrity, authentication, and non-repudiation[3].

Typically, a cryptosystem is being consisted of three algorithms: one for key generation, one for encryption, and one for decryption[4]. The data confidentiality aspect points to the encryption and decryption algorithms while the aspects data integrity, authentication and non-repudiation is counter-parted by the key generation algorithm. Within these three algorithms lies an algorithm that also supports the aspects of data integrity and authentication requirements. This is the hash function which helps in mapping data during encryption and decryption[5] and the cryptographic hash function which provides cryptographic hash codes or message digest that can be used for authentication and as basis for data integrity[6].

According to Stephen Northcutt (2002), there are three types of cryptography algorithms: secret key, public key, and hash functions[7]. However, unlike secret key and public key algorithms, hash functions, which are also called message digests or one-way encryption, do not implement a key, instead, a fixed-length hash value is computed based on the plaintext that makes it impossible [or computationally infeasible[8]] for either the contents or length of the plaintext to be recovered. The hash value provides a digital fingerprint of a message's contents, which ensures that the message has not been altered by an intruder, virus, or by other means[9].

There is a long list of cryptographic hash functions. Two of the most commonly used cryptographic hash functions are MD5 and SHA-1. MD5 or Message Digest number 5 was invented by Ron Rivest in 1992 as an improved version of MD4 he created in 1990. Meanwhile, in 1993, the National Security Agency published a hash function very similar to MD5, called the Secure Hash Algorithm (SHA). Then in 1995, citing a newly discovered weakness that it refused to elaborate on, the NSA made a change to SHA. The new algorithm was called SHA-1. Today, the most popular hash function is SHA-1[8], with MD5 still being used.

To understand the viability of using hash functions to verify integrity and source of information, one must first examine the properties and origin of the basic hash function. The objective of this study is to analyze the algorithms, strengths and weaknesses of the 2 most commonly used modern cryptographic hash functions: MD5 and SHA-1. It shall evaluate the performance of the compared hash functions in terms of their speed to generate hash value over a relative file size. It shall also gather data on how it would be computationally infeasible to Brute Force the hash algorithms. By examining the history and security available in each function, it aims to benefit application and web developers to determine which timely algorithm is best suited for his software development endeavors.

MATERIALS AND METHOD

Hash Functions: An Overview

Hash functions are mathematical functions that take in a relatively arbitrary amount of data as an input, also called as the hash key, pre-image or message, and produce an output of fixed size which is also called the hash, hash value, hash code, or message digest. The output is always the same when given the same input. Any piece of data can be used as a hash key such as character strings, binary files, website address, or computer network packets, and subject it to a hash function to produce fixed length hash value.

If two different hash key produces the same hash value, such occurrence is called a collision which is avoided, as much as possible, in order to attain a one-is-to-one relationship of hash key and hash value – that is every hash key must have and must be linked its own unique hash value.

If a hash function produces hash values according to a specified fixed length string format or pattern, such hash function is usually used as a hash table. However, if it assigns a fixed length

random string, such hash function is called cryptographic hash function which is usually used for data integrity evaluation and data authentication by taking into consideration that since different hash keys almost always produce different hash values when subjected to a hash function, if a hash value of an input hash key changes, then the input hash key itself has changed.

Properties of Hash Functions

The first main property of a hash function, which all hash functions have, is *one-wayness* or being a one-way function[4]. Given knowledge of an input hash key x, we can easily compute the hash value $h(x)$ using the hash function; but it is very difficult given the hash value $h(x)$ to find a corresponding pre-image x if one is not already known. As the output is random, the best an attacker who wants to invert a random function can do is to keep on feeding in more inputs until he gets lucky. This property provides computational hardness to a hash function.

A second property of hash functions is that the hash value $h(x)$ will not give any information at all about even a part of the input hash key x[2]. To make a backward hash attack more difficult, some hash functions use a secret key or salt k through concatenation and computing $h(x, k)$[8]. This ensures that an attacker will have a hard time identifying which of his decrypted hash value is actually the real hash key as it contains some characters which is not originally part of the hash key.

A third property of hash functions is the difficulty to find collisions[9] by taking into considerations that different messages $M_1 \neq M_2$ with $h(M_1)=h(M_2)$. If an attacker finds a shortcut attack, it only means that the hash function is not really using a good pseudorandom sub-function within it, otherwise, the best way for an opponent of finding a collision is to collect a large set of messages M_i and their corresponding hash values $h(M_i)$, sort the hashes, and look for a match. A hash function with an n-bit number output would have 2^n possible hash values, as such, the number of hashes an attacker will need to compute before he can find a match will be about $2^{n/2}$ hash values. This fact is of major significance in computer security.

Popular Cryptographic Hash Functions

There are two primarily cryptographic hash functions in use today, MD5 and SHA-1[9].

MD5 Hash Function

MD5 stands for “Message Digest 5” as it is the fifth revision of a message digest algorithm devised by Ron L. Rivest of RSA Laboratories. The early revisions of this algorithm were published prior to 1989, and the most recent revision of the algorithm was published in 1991. It has an arbitrary input length and produces a 128-bit digest. Although weaknesses have been found in the algorithm[8], there has never been a published collision as of this writing.

In the MD5 algorithm, the input message is broken up into chunks of 512-bit blocks (sixteen 32-bit words) and being padded so that each chunk’s length is divisible by 512. Each message block is processed using the main MD5 Function (F) which operates on a 128-bit state, divided into four 32-bit words, denoted A, B, C, and D. Function F is composed of four similar stages called rounds. Each round is composed of 16 similar operations based on a non-linear function F, modular addition, and left rotation. Figure 1 illustrates the operations in a round.

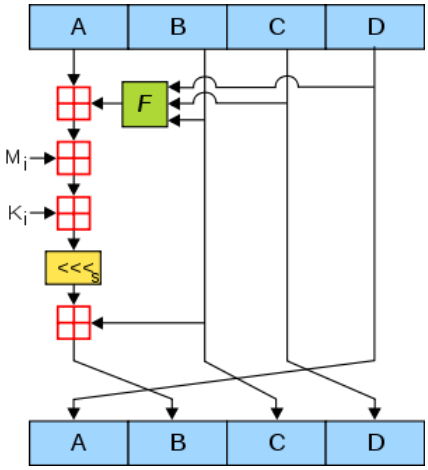

Figure 1. The Main MD5 Function.

SHA1 Hash Function

SHA1 stands for "Secure Hash Algorithm 1", it is the first revision of a hash algorithm developed by the National Security Agency of the United States of America. The algorithm was first published in 1995. SHA1 supports messages of any length less than 264 bits as input, and produces a 160-bit digest. In the unlikely event that one wishes to compute the digest of a message larger than 264 bits in length (over 2 billion Gigabytes of information), the simplest solution would be to divide the large messages into smaller messages. There are no known weaknesses in SHA1, and it is generally considered the more secure of the two algorithms. There are also variations of SHA1 which produce longer digests, SHA-256, SHA-512. They produce digests of 256 bits and 512 bits, respectively[10].

Figure 2 shows one iteration within the SHA1 compression function which has 5 chunks of 32-bit words (as compared to MD5 which has only 4) where the first 2 chunks will undergo bit rotation, the 3 middle chunks shall be processed under a varying non-linear function, and the last chunk, together with the outputs of first 5 chunks, will undergo addition modulo 2^{32}. Chunk swapping (first to next chunk and last to first chunk) will be processed in the last stage.

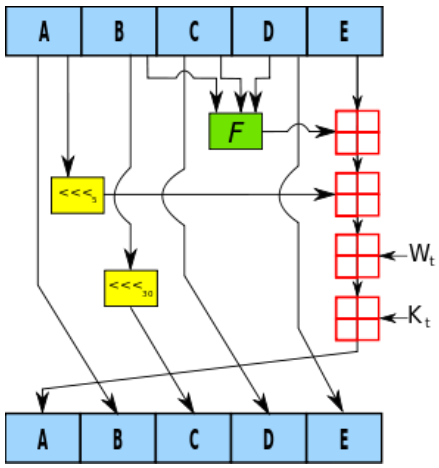

Figure 2. The SHA1 Algorithm.

The SHA1 and MD5 algorithms are considered secure because there are no known techniques to find collisions, except via brute force. In a brute force attack random inputs are tried, storing the results until a collision is found. If we do not limit ourselves to finding a collision with a specific

message, one can expect to find a collision within $2^{n/2}$ computations, where n is the number of bits in the digest. This is commonly known as the birthday attack[11]. This means that an attacker would need to compute the digests of approximately 264 messages to find a collision in the MD5 function, and approximately 280 computations to find a collision in SHA1. Note that SHA1 may be more secure than MD5, but it is more costly to compute a message digest using SHA1 than MD5. If one is expressing security concerns SHA1 would be the function of choice, however, if speed is an issue it is likely that MD5 would result in faster performance, and would likely still be secure enough for most applications. In August 2001, a complex computing grid theorized by IBM was believed to be able to achieve 13.6 trillion calculations per second, which would make it one of the most powerful computers known.

Even at this rate, assuming one computation of a digest per super computer calculation, it would take over 2,800 years to find a collision in SHA1. In the unlikely event that a collision was ever found, security minded individuals could just use one of the SHA algorithms that produce larger outputs; these algorithms would require an even greater amount of time to find collisions in.

Comparative Analysis Setup

The evaluations in this paper were conducted in to two ways: performance evaluation and computational hardness evaluation. The performance evaluation assessed the speed of the two cryptographic algorithms in producing hash values over a set of files with varying file sizes using a VB.NET tool created by the researchers using Microsoft Visual Studio 2010. Meanwhile, the computational hardness evaluation calculated the difficulty to decipher each subject algorithm using Brute Force process, a cryptanalytic attack that guesses all possible combinations of characters or keys for a specified string length[12].

To simulate the performance of MD5 and SHA-1, this study used the available security and string classes of Microsoft VB.NET under Microsoft Visual Studio 2010 Integrated Development Environment (IDE). String classes are used to generate random string and for string comparison. The ASCII character set was selected as the character encoding of the output hash values which uses 1 byte per character. The simulation tool is compiled to support version 4.0 of the Microsoft .NET framework. The implementation is thoroughly tested and is optimized to give the maximum performance for the two cryptographic hash functions.

Table 1 displays the input parameters used in this study in evaluating the performance of each cryptographic hash algorithm over a set of digital computer files.

Table 1. Key or Salt input parameters of compared cryptographic hash functions in evaluating speed of producing cryptographic hash values over computer files.

Key/Salt Parameters for String Evaluation

Cryptographic Hash Function	Key / Salt
MD5	FHERn@ndosvir@yjr808
SHA-1	FHERn@ndosvir@yjr808

To evaluate the computational hardness of each hash algorithm, this paper utilized an online Brute Force Calculator (calc.opensecurityresearch.com) and set the following as input parameters:

```
password length: 7 to 20 characters long

character set : a..z, A..Z, 0 to 9, and !@#$%^&*()-
```

```
_+=~`[]{} |\:;"'<>,.?
```

Table 2 details Keys Per Second (kps) used in this study to evaluate the computational hardness of each subject cryptographic hash algorithm aside from the input parameters stated above. The values are determined according to the speed of the computer unit used by openresearch.com's online Brute Force Calculator.

Keys per Second Input Parameters

Cryptographic Hash Function	Keys per Second (kps)
MD5	7,508,000
SHA-1	7,107,000

Table 2. Keys per Second input parameters used in utilizing opensearch.com's online Brute Force Calculator to evaluate computational hardness of each compared hash algorithm.

Comparative Analysis Methodology

The specifications of the computer unit used in the simulations are as follows:

Processor : 2.20GHz Quad Core Intel Core i3

Memory : 4 Gigabytes

Operating System : Windows 8 64-bit

The default settings in VB.NET of Microsoft Visual Studio 2010 were used as parameters in compiling the simulation program. To assure that the results are consistent, the simulations are performed for several times. The following image, Figure 3, shows the screenshot of the actual VB.NET tool created.

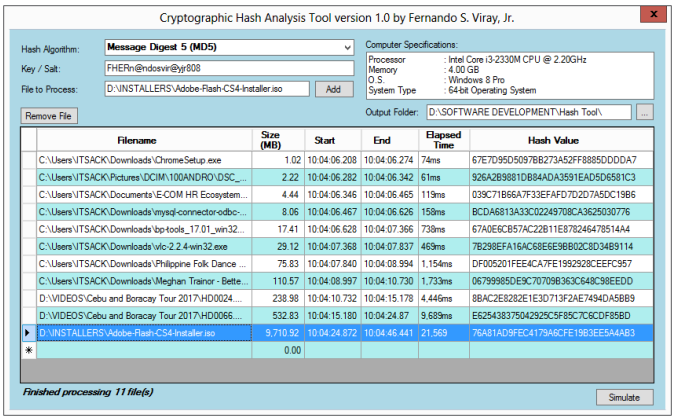

Figure 3. Screenshot of the actual VB.NET tool created by the researchers using MS Visual Studio 2010 where MD5 and SHA-1 cryptographic hash algorithms were implemented using the IDE's built-in Cryptography classes.

The evaluation of the performance of each compared cryptographic hash function in terms of speed, a set of random files of varying sizes (from 1 Megabyte to 1 Gigabyte) were subjected to each algorithm's hash function and recorded the time in ticks since the function easily finished even before a millisecond has elapsed with the specifications of the computer used. A single tick represents one ten-millionth of a second or one hundred nanoseconds, this means that there are 10,000 ticks in a millisecond, or 10 million ticks in a second[13].

Since MS-Visual Studio 2010 already has Cryptography classes where MD5 and SHA-1 implementation are included, the following function pseudo code was used as basis of the VB.NET tool created by the researchers to get the exact time how fast each hash function generates hash value after subjecting a file to the hash function.

```
Function GetHashGenerationTime(sFile() as Byte,
password as String) as Long Integer;
  begin
    timeStart = GetCurrentTimeInTicks();
    hashValue = CryptClass.GetHash(sFile,password);
    timeEnd = GetCurrentTimeInTicks();
    timeElapsed = timeEnd – timeStart;
    return timeElapsed;
  end
```

To convert the number of ticks covered by the hash function in generating hash values into seconds, the following formula is used:

$$time_{seconds} = \frac{time_{ticks}}{10{,}000 \text{ ticks/second}}$$

To evaluate the computational hardness of each algorithm by using Brute Force, the researchers used an online Brute Force Calculator (calc.opensecurityresearch.com) which calculates the time it would take in providing all possible keys or character combinations taking into consideration the password length, keys per second and character set as input parameters. A 3.4GHz Intel Core i7-2600K computer was used by the online Brute Force Calculator and calculated the projected computation time with the following formula:

$$t_{years} = \frac{l^c}{k}$$

where:

t = the projected Brute Force time in years

l = length of password

c = number of characters in the character set

k = the keys per year capability of a computer

Because k is usually expressed in keys per second (kps), below is the formula used in converting keys per second to keys per year:

$$k_{years} = k_{seconds} \times 60\text{sec/min} \times 60\text{min/hr} \times 24\text{hr/day} \times 365.25\text{days/yr}$$

or

$\mathbf{k_{years}} = k_{seconds}$ x 31,557,600seconds/year

RESULTS AND DISCUSSION

Analysis Results

Figure 4 below shows the results of the performance evaluation of MD5 and SHA-1 in terms of how long the algorithm generated a cryptographic hash value of the random files. It exhibits the superiority of MD5 algorithm and how it outperforms SHA-1 in terms of processing time especially on relatively large files. It also shows that SHA-1 consumes more computing resources as input file size becomes relatively larger.

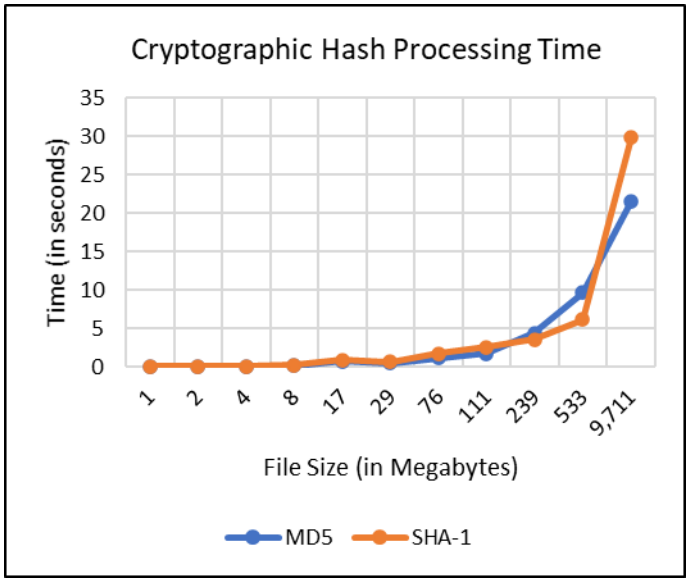

Figure 4. Performance Results of MD5 and SHA-1 algorithms in terms of Hash Value Generation.

Table 3 displays the Brute Force results of MD5 and SHA-1 using the online Brute Force Calculator of opensecurityresearch.com. It suggests that passwords should be at least 8 characters long in order to provide hardness in brute forcing. It also indicates that a Brute Force attack on SHA-1 algorithm would take a huge amount of time and processing power as compared to MD5.

Brute Forcing MD5 and SHA-1
(in years)

Password Length	MD5	SHA-1
7	0.296499657	0.31297186
8	28.29924502	29.89773507
9	2688.370625	2840.057653
10	255395.269	269805.4997
11	24262550.62	25631522.45
12	2304942309	2434994633
13	2.1897E+11	2.31324E+11
14	2.08021E+13	2.19758E+13
15	1.9762E+15	2.0877E+15

16	1.87739E+17	1.98332E+17
17	1.78352E+19	1.88415E+19
18	1.69434E+21	1.78994E+21
19	1.60963E+23	1.70045E+23
20	1.52915E+25	1.61543E+25

Table 3. Projected Brute Force attack time of MD5 and SHA-1 algorithms using openresearch.com's Online Brute Force Calculator.

CONCLUSION

The Message Digest number 5 (MD5) algorithm has a better performance in terms of speed in generating hash value as compared to Secure Hash Algorithm 1 (SHA-1). However, SHA-1 has more computational hardness under a Brute Force attack as compared to the MD5 algorithm. Results also prove that it would take a very long time before the two cryptographic hash algorithms can be Brute Forced or it would need a huge number of computers which will perform simultaneous parallel brute forcing before the compared algorithms can be cracked.

REFERENCE

[1] Liddell, Henry George; Scott, Robert; Jones, Henry Stuart; McKenzie, Roderick (**1984**). *A Greek-English Lexicon*. Oxford University Press.

[2] Biggs, Norman (**2008**). *Codes: An introduction to Information Communication and Cryptography*. Springer. p. 171.

[3] Menezes, A. J.; van Oorschot, P. C.; Vanstone, S. A. *Handbook of Applied Cryptography*. ISBN 0-8493-8523-7. Archived from the original on 7 March **2005**.

[4] Buchmann, Johannes A. (**2004**) *Introduction to Cryptography (2nd ed.)*. Springer. ISBN 0-387-20756-2.

[5] Konheim, Alan (**2010**). "7. HASHING FOR STORAGE: DATA MANAGEMENT". *Hashing in Computer Science: Fifty Years of Slicing and Dicing*. Wiley-Interscience. ISBN 9780470344736.

[6] Sedgewick, Robert (**2002**). "14. Hashing". *Algorithms in Java (3 ed.)*. Addison Wesley. ISBN 978-0201361209.

[7] Northcutt, Stephen (**2002**). Network Intrusion Detection (3rd ed.). Sams Publishing. ISBN 978-0735712652

[8] Schneier, Bruce (**2004**). *"Cryptanalysis of MD5 and SHA: Time for a New Standard"*. Computerworld.

[9] Silva, John .Edward. (**2003**). *An Overview of Cryptographic Hash Functions and Their Uses*. SANS Institute InfoSec Reading Room. SANS Institute.

[10] Eastlake, Donald E. and Jones Paul E. (2001). *US Secure Hash Algorithm (SHA1)*, Internet RFC 3174 September **2001**.

[11] Halevi, Shai and Krawczyk, Hugo (**2006**). *Strengthening Digital Signatures via Randomized Hashing*. IBM T.J. Watson Research Center, Yorktown Heights, New York.

[12] Paar, Christof; Pelzl, Jan; Preneel, Bart (**2010**). *Understanding Cryptography: A Textbook for Students and Practitioners*. Springer. ISBN 3-642-04100-0.

[13] Microsoft Developer Network. Article: *DateTime.Ticks Property*.

https://msdn.microsoft.com/en-us/library/ system.datetime.ticks(v=vs.110).aspx, accessed **2017**-03-09.