ISSN:0975-9646

Performance Analysis of Hash Algorithms and File Integrity

Hanumantu Rajeswari [#1], Ramesh Yegireddi [#2], Vudumula Govinda Rao[#3]

[12] *Department of CSE*
[1]*Aditya Institute of Technology and Management Tekkali - 532201,*
Andhra Pradesh, India

[3] *Department of CSE*
Swami Vivekananda Engineering College, Bobbili,
Andhra Pradesh, India

***Abstract*— This paper is concerned with giving both an overview of the performance analysis of hash functions in cryptography and a presentation of file integrity in mobile phones.**
Cryptographic hash functions are a very useful tool in cryptography. They are applied in many areas like integrity of messages, storage of passwords securely and protect signatures. The three hash algorithms SHA-1, SHA-512 and MD5 are considered to analyze their performance.
Hash functions are widely used to verify file integrity. And, it is clear that the message digest is used to verify the integrity of the document. Indeed, it certifies that the document has not been modified somewhere between the moment it was sent and the moment it was received.
We analysed the performance of three algorithms on 32 bit processor and identified the SHA-1 has best performance; the same algorithm is used in file integrity.

***Keywords*— message digest, hash value, SHA, integrity, cryptography, performance.**

I. INTRODUCTION

A hashing algorithm is a deterministic function that takes in an arbitrary length block of data, and returns a fixed-size string, which is called the message digest value.[1] The digest is sometimes also called the "hash" or "fingerprint" of the input.

A hash function H is a transformation that takes a variable-size input "m" and returns a fixed-size string, which is called the hash value h (that is, h = H(m)).[2] Hash functions with just this property have a variety of general computational uses, but when employed in cryptography the hash functions are usually chosen to have some additional properties.

The basic requirements for a cryptographic hash function are:

- The input can be of any length,
- The output has a fixed length,
- H(x) is relatively easy to compute for any given x ,
- H(x) is one-way,
- H(x) is collision-free.

A hash function H is said to be one-way if it is hard to invert, where "hard to invert" means that given a hash value h, it is computationally infeasible to find some input x such that H(x) = h.

If, given a message x, it is computationally infeasible to find a message y not equal to x such that H(x) = H(y) then H is said to be a weakly collision-free hash function.

A strongly collision-free hash function H is one for which it is computationally infeasible to find any two messages x and y such that H(x) = H(y).

II. COMMON USES OF CRYPTOGRAPHIC HASHES

A. *Integrity Verification*
The sender can hash a file and appended that message digest before sending to the recipient. The recipient will then hash the file received and check the hashes match, to ensure files have not been corrupted or modified.

B. *Digital signature protection*
A hash is generated, signed with private key of sender and transmitted with its message, allowing the recipient to hash the message and decrypt the digest using sender's public key to compare outputs. By signing the hash before sending, the sender can prove that the message has not been tampered with.

C. *Passwords Protection*
Rather than storing a user's password, a system will typically store the hash of the password instead. When a user enters their password, the hash is then computed and compared with the stored hash. If the hash matches, due to the collision resistance property of hashing algorithms, it implies that the passwords match.

III. STRUCTURE OF HASH ALGORITHMS

A. *SHA-1 algorithm*

The SHA-1 algorithm [3] accepts as input a message with a maximum length of 2^{64} - 1 and produces a 160-bit message digest as output. The message is processed by the compression function in 512-bit block. Each block is divided further into sixteen 32-bit words denoted by M_t for t = 0, 1. . . 15. The compression function consists of four rounds; each round is made up of a sequence of twenty steps. A complete SHA-1 round consists of eighty steps where a block length of 512 bits is used together with a 160-bit chaining variable to finally produce a 160-bit hash

value. The processing works as described in the following steps:

Step 1: Append padding bits

The original message is padded so that its length is congruent to 448modulo512. Again, padding is always added although the message already has the desired length. Padding consists of a single 1 followed by the necessary number of 0 bits.

Step 2: Append length

A 64-bit block treated as an unsigned 64-bit integer (most significant byte first), and representing the length of the original message (before padding in step 1), is appended to the message. The entire message's length is now a multiple of 512.

Step 3: Initialize the buffer

The buffer consists of five (5) registers of 32 bits each denoted by A, B, C, D, and E.

This 160-bit buffer is used to hold temporary and final results of the compression function. These five registers are initialized to the following 32-bit integers (in hexadecimal notation).

A = 67 45 23 01
B = ef cd ab 89
C = 98 ba dc fe
D = 10 32 54 76
E = c3 d2 e1 f0

Step 4: Process message in 512-bit blocks

- Divide Mi into 16 32-bit words: W0, W1, W2, . . . , W15.
- for t = 16 to 79 compute
- Wt = (Wt−3 $\oplus$ Wt−8 $\oplus$Wt−14 $\oplus$Wt−16) <<< 1.
- Set (A0, B0, C0, D0, E0) ← hi−1.
- For t = 0 to 79 do
  - T = At <<< 5 + ft(Bt , Ct ,Dt) + Et +Wt + Kt
  - Et+1 = Dt , Dt+1 = Ct , Ct+1 = Bt <<< 30, Bt+1 = At , At+1 = T.

Step 5: Output

After processing the last 512-bit message block, we obtain a 160-bit message digest.

- Output A = A0 + A80, B = B0 + B80, C = C0 + C80, D = D0 + D80, and E = E0 + E80 (modulo 2^{32}).
- The function ft and the values Kt used above are:
  - $0 \le t \le 19$: ft (X, Y , Z) = XY $\vee$ ($\neg$X)Z Kt = 5A827999
  - $20 \le t \le 39$: ft (X, Y , Z) = X $\oplus$ Y $\oplus$ Z Kt = 6ED9EBA1
  - $40 \le t \le 59$: ft (X, Y , Z) = XY $\vee$ XZ $\vee$ YZ Kt = 8F1BBCDC
  - $60 \le t \le 79$: ft (X, Y , Z) = X $\oplus$ Y $\oplus$ Z Kt = CA62C1D6

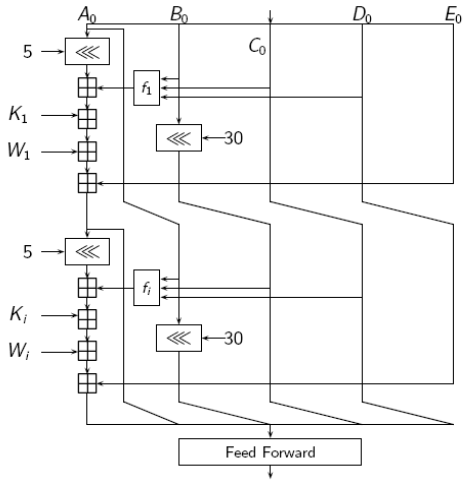

Fig. 1 SHA – 1 processing

B. *MD5 Algorithm*

Ronald Rivest's MD5 function is a cryptographic algorithm that takes an input of arbitrary length and produces a *message digest* that is 128 bits long. MD5 is used in many situations where a potentially long message needs to be processed and/or compared quickly. The most common application is the creation and verification of digital signatures.

The MD5 algorithm first divides the input in blocks of 512 bits each. 64 Bits are inserted at the end of the last block. These 64 bits are used to record the length of the original input. If the last block is less than 512 bits, some extra bits are 'padded' to the end. Next, each block is divided into 16 words of 32 bits each. These are denoted as M0 ... M15.

1). The buffer

MD5 uses a buffer that is made up of four words that are each 32 bits long. These words are called A, B, C and D. The initial value is:

A = 67452301
B = EFCDAB89
C = 98BADCFE
D = 10325476

2). The table

MD5 further uses a table K that has 64 elements. Element number i is indicated as Ki. The table is computed beforehand to speed up the computations. The elements are computed using the mathematical sin function:

$$Ki = | \sin(i + 1)| * 2^{32}$$

3). Four auxiliary functions

In addition MD5 uses four auxiliary functions that each take as input three 32-bit words and produce as output one 32-bit word. They apply the logical operators and, or, not and xor to the input bits.

F(X,Y,Z) = (X $\wedge$ Y) or (($\neg$X) $\wedge$ Z)
G(X,Y,Z) = (X $\wedge$ Z) or (Y $\wedge$ ($\neg$Z))
H(X,Y,Z) = X $\oplus$Y $\oplus$Z
I(X,Y,Z) = Y $\oplus$(X $\vee$ ($\neg$Z))

4). Processing the blocks

The contents of the four buffers (A, B, C and D) are now mixed with the words of the input, using the four auxiliary functions (F, G, H and I). There are four *rounds*, each involves 16 basic *operations*. One operation is illustrated in the figure below.

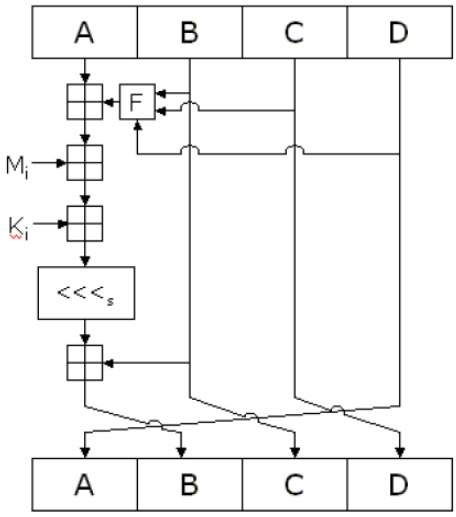

Fig. 2 MD 5 Processing

The figure shows how the auxiliary function F is applied to the four buffers (A, B, C and D), using message word Mi and constant Ki. The item "<<<s" denotes a binary left shift by *s* bits.

5). The output

After all rounds have been performed, the buffers A, B, C and D contain the MD5 digest of the original input.

C. *SHA – 512 Algorithm*

For **SHA-512** algorithm [5], each message block has **1024 bits**, which are represented as a sequence of sixteen **64-bit words**.

1). SHA-512 Constants

SHA-512 use sequence of eighty constant 64-bit words, $K_0^{\{512\}}$, $K_1^{\{512\}}$, ... , $K_{79}^{\{512\}}$. These words represent the first sixty- four bits of the fractional parts of the cube roots of the first eighty prime numbers. In hex, these constant words are (from left to right)

428a2f98d728ae22	7137449123ef65cd
b5c0fbcfec4d3b2f	e9b5dba58189dbbc
3956c25bf348b538	59f111f1b605d019
923f82a4af194f9b	ab1c5ed5da6d8118
d807aa98a3030242	12835b0145706fbe
243185be4ee4b28c	550c7dc3d5ffb4e2
72be5d74f27b896f	80deb1fe3b1696b1

(Sample data)

2). Padding

Suppose the length of the message *M*, in bits, is ll bits. Append the bit "1" to the end of the message, followed by *k* zero bits, where *k* is the smallest non-negative solution to the equation l □□1 □□*k* ≡□896mod1024.

SHA-512 use six logical functions. E*ach function operates on 64-bit words* and is represented as *x*, *y*, and *z*. The result of each function is a new 64-bit word.

$$Ch(x, y, z) = (x \Lambda \square y) \oplus \square (x \Lambda \square z)$$

$$Maj(x, y, z) = (x \Lambda \square y) \oplus \square (x \Lambda \square z) \oplus \square (y \Lambda \square z)$$

$$\sum_0^{\{512\}}(x) = ROTR^{28}(x) \oplus ROTR^{34}(x) \oplus ROTR^{39}(x)$$

$$\sum_1^{\{512\}}(x) = ROTR^{14}(x) \oplus ROTR^{18}(x) \oplus ROTR^{41}(x)$$

$$\sigma_0^{\{512\}}(x) = ROTR^{1}(x) \oplus ROTR^{8}(x) \oplus SHR^{7}(x)$$

$$\sigma_1^{\{512\}}(x) = ROTR^{19}(x) \oplus ROTR^{61}(x) \oplus SHR^{6}(x)$$

IV. PERFORMANCE COMPARISON

The comparison of three hash algorithms is shown in the table 1. The output hash values of the three algorithms also show in the table 2. The performance of these algorithms verified on 32-bit processor, the SHA – 1 has better performance as shown in table 3.

Name	Block Size (Bits)	Word Size (Bits)	Output Size (Bits)	Rounds	Length Field (Bits)
MD 5	512	32	128	64	64
SHA – 1	512	32	160	80	64
SHA – 512	512	32	512	80	128

Table 1 : Comparison of Three Algorithms

Name	Input String	Hash Value (Message Digest)	Size of the MD
MD 5	Aditya Institute of Technology and Management	*e871b93a38d8a528bfd5d421aebfdcas*	*128*
SHA –1	Aditya Institute of Technology and Management	*0ed1022d39b7471dd91d61f713d1b54e15115175*	*160*
SHA–512	Aditya Institute of Technology and Management	*0df1451djf14512ddbbf1171ddiba012457dfeib115432eedfa2173ffdeb112f2idb1115412*	*512*

Table 2 : Hash Value of Three Algorithms

Algorithm	Performance (ms)
MD5	11.027356465389207
SHA – 1	10.275190830230713
SHA - 512	18.339390993118286

Table 3 : Performance of Three Algorithms

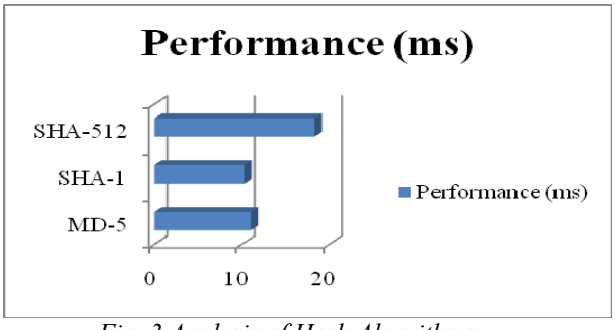

Fig. 3 Analysis of Hash Algorithms

V. FILE INTEGRITY

There are some possibilities to enhance the integrity of files in Android Mobile. We propose two possibilities in this regard which require a lot of work to be done on these two alternatives. According to our proposal, (1) to enhance file integrity, we need SHA -1 algorithm. When the mobile is equipped with SHA - 1 algorithm, any modified file can't be restored in the mobile. This SHA - 1 Algorithm can be applied to all contents of the mobile or any specific content of the mobile. As we confine to file fabrication and restoration, we propose to apply the SHA - 1 Algorithm specifically to the contents of android mobile. Immediately after selecting the USB debugging option, a unique Hash Value for every file in the android mobile is generated. Those Hash values must be stored in permanent memory of the Android Mobile. As we all know that these Hash values cannot be changed and these values will be in the permanent memory until the USB Debugging is disabled. We propose that the Hash value of the original file should not go along with the copy of the targeted file, when it is exported in Xml format [6] to the PC. The fabricated file is sent through Data transfer Protocols, and it has a new Hash value which is generated while it is being imported to the mobile. The Hash value of the fabricated file must be stored in other than permanent memory of the Android mobile. Now the new Hash value must be compared to the Hash values which are in the permanent memory of the Android mobile. When the Hash value of the fabricated file is not matched with any one of the Hash values, the mobile does not accept it. In this process, we may prevent the file from being fabricated.

We wish to propose another alternative which is very simple. According to our proposal (2), the exported file which is usually sent through Xml format to the PC. The targeted file is opened and fabricated. To make the targeted file unchangeable, the (exported) Xml file must be made as only readable when it is being exported onto the PC.

VI. CONCLUSION

We compared and analysed three algorithms MD5, SHA-1 and SHA-512. The SHA-1 has best performance on 32-bit processor. We also proposed a novel and efficient way of storing files in Android Mobile devices such that if any file is taken as evidence in the court of law. SHA-1 is not patented. It may be used free of charge for any purpose .With the help of SHA-1 Algorithm we can ensure the integrity of files.

REFERENCES

[1]. www.asd.gov.au/publications/csocprotect/SHA-1/ Deprecated
[2]. Significance of Hash Value Generation in Digital Forensic: A Case Study, IJERD, Volume 2, Issue 5 (July 2012), PP. 64-70
[3]. Cryptography and Network Security, Fifth Edition, William Stallings Prentice Hall 2010,ISBN-10: 0136097049
[4]. http://www.iusmentis.com/technology/hashfunctions/md5
[5]. http://csrc.nist.gov/cryptval/512
[6]. Enhancing the Integrity of Short Message Service, IJCSI, Vol. 10, Issue 6, No 2, November 2013