Algoritmos y Estructuras de Datos II

Grupo 1

Tarea Extraclase 4

Profesor:

Luis Diego Noguera Mena

Integrantes:

Johnny Agüero Sandí

Ignacio Morales Chang

10/6/2021

Contents

Introduction

A hash function is a mathematical function that converts a numerical input value of arbitrary length into a compressed numerical value of fixed length. These values returned are called message digest. A hash function is desired to have the following properties:

- Pre-image resistance: it should be computationally hard to reverse a hash function.
- Second pre-image resistance: given an input and its hash, it should be hard to find a different input with the same hash.
- Collision resistance: it should be hard to find to find two different inputs of any length that result in the same hash, also referred as collision free hash function.

Some popular hash functions: Message Digest, Secure Hash Function, RIPEMD, and Whirlpool. In this document one of the Message Digest and one of the Secure Hash Function will be described.

Hash functions have two direct applications bases on its cryptographic properties: password storage and Data integrity check.

Problem Analysis

In this evaluation, the main objective is that the students comprehend what are hash functions and some of their appliances. To resolve this, investigation must be done by the students. One of the functions to be found of hash functions is data integrity check. Data integrity check is important in the modern world since many important files are shared in the digital form. Data integrity check protects the validity and accuracy of data. It also guarantees the searchability a traceability of your data to its original source.

SHA1

SHA1 is a Secure Hash Algorithm for computing a condensed representation of a message or data file. A 160-bit output, called message digest, is produced by SHA-1 when a message of any length < 2^64 bits is input, then, the message digest can be input to a signature algorithm that generates or verifies the signature for the message. Doing this is more efficient when compared to signing the message instead of the message digest, this is due to the much smaller size of the message digest. This same algorithm must be used by the verifier of a digital signature as was used by the creator of the digital signature. Any change to the message in transit will most likely result in a different message digest, resulting on failing to verify the signature.

Definitions:

- Hex Digit: Representation of a 4-bit string composed of elements of the set {0, 1, …, 0, A, B, …, F}.
- To convert a word (32-bit string), each 4-bit string is converted to its hex equivalent.

- An integer between 0 and 2^32-1 inclusive may be represented as a word. Example: 291 = 2^8 + 2^5 + 2^1 + 2^0 = 256 + 32 + 2 + 1 ---> hex = 00000123. If, for example, the integer z, where 0 <= z < 2^64 then z = (2^32)x + y being z < 2^32 and 0 <= y < 2^32. “z” can be represented as the pair of words (X, Y).
- Block = 512-bit string. It may be represented by a sequence of 16 words.

Message Padding:

SHA-1 is used to compute a message digest for a message or data file that is provided as input. This input should be considered as a bit string, where the length is the number of bits in the message. When the length is a multiple of 8, it can be represented as its hex equivalent. The purpose of number padding is to make the total length of a padded message a multiple of 512. A “1” followed by m “0”s followed by a 64-bit integer are appended to the end of the message to produce a padded message of length 512*n. The 64-bit integer is the length of the original message. Now, the padded message will be processed by SHA-1 as n 512-bit blocks.

Functions used:

f(t;B,C,D) = (B AND C) OR ((NOT B) AND D) (0 <= t <= 19)

f(t;B,C,D) = B XOR C XOR D (20 <= t <= 39)

f(t;B,C,D) = (B AND C) OR (B AND D) OR (C AND D) (40 <= t <= 59)

f(t;B,C,D) = B XOR C XOR D (60 <= t <= 79).

Constants used:

K(t) = 5A827999 (0 <= t <= 19)

K(t) = 6ED9EBA1 (20 <= t <= 39)

K(t) = 8F1BBCDC (40 <= t <= 59)

K(t) = CA62C1D6 (60 <= t <= 79).

There are different methods to yield the same message digest, we will be covering one of them. In this method the message digest is computed using the message padded. The compution is described using two buffers, each of five 32-bit words and a sequence of eighty 32-bit words. The words on the first 5-word buffer are labeled A, B, C, D, E whereas the words of the second 5-word buffer are labeled H0, H1, H2, H3, H4. The words of the 80-word sequence are labeled W(0), W(1), …, W(79).

The 16-word block M(1), M(2), …, M(n) defined when message padding, are processed in order. Before processing anything, the H’s are initialized as:

H0 = 67452301

H1 = EFCDAB89

H2 = 98BADCFE

H3 = 10325476

H4 = C3D2E1F0.

Now M(1), M(2), ... , M(n) are processed. To process M(i), we proceed as follows:

a. Divide M(i) into 16 words W(0), W(1), ... , W(15), where W(0) is the left-most word.
b. For t = 16 to 79 let
   W(t) = S^1(W(t-3) XOR W(t-8) XOR W(t-14) XOR W(t-16)).
c. Let A = H0, B = H1, C = H2, D = H3, E = H4.
d. For t = 0 to 79 do
   TEMP = S^5(A) + f(t;B,C,D) + E + W(t) + K(t);
   E = D; D = C; C = S^30(B); B = A; A = TEMP;
e. Let H0 = H0 + A, H1 = H1 + B, H2 = H2 + C, H3 = H3 + D, H4 = H4 + E.

After being processed, the message digest is the 160-bit string represented by:

H0 H1 H2 H3 H4

MD5

Algorithm designed by Ron Rivest (author of RC-2, RC-4 and RC-5 and one of the co-authors of the RSA). They produce a 128-bit digest of the message. Designed as an improvement on the MD4 algorithm, with respect to which it is slower; but more secure. Inspired by the SHA1 algorithm.

The algorithm takes as input a message of arbitrary length and outputs a "fingerprint" or "message digest" of the input. It is conjectured that it is computationally infeasible to produce two messages that have the same message digest, or to produce any message that has a given prespecified destination message digest. The MD5 algorithm is designed for digital signature applications, where a large file must be securely "compressed" before being encrypted with a private (secret) key under a public key cryptosystem such as RSA.

Definitions:

- Word: is a 32-bit quantity and Byte is an eight-bit quantity.
- A sequence of bytes can be interpreted as a sequence of 32-bit words.
- “x sub I”: expression that indicates that i is a subscript of x.
- “x^I”: denotes x to the i-th power.
- Block = 512-bit string. It may be represented by a sequence of 16 words.

Message Padding:

We begin by supposing that we have a b-bit message as input, and that we wish to find its message digest. Here b is an arbitrary nonnegative integer; b may be zero, it need not be a multiple of eight, and it may be arbitrarily large. We imagine the bits of them message written down as follows:

The message is "padded" (extended) so that its length (in bits) is congruent to 448, modulo 512. That is, the message is extended so that it is just 64 bits shy of being a multiple of 512 bits long. Padding is always performed, even if the length of the message is already congruent to 448, modulo 512.

Append Length:

A 64-bit representation of b (the length of the message before the padding bits were added) is appended to the result of the previous step. In the unlikely event that b is greater than 2^64, then only the low-order 64 bits of b are used. (These bits are appended as two 32-bit words and appended low-order word first in accordance with the previous conventions.).

Initialize MD Buffer:

A four-word buffer (A,B,C,D) is used to compute the message digest. Here each of A, B, C, D is a 32-bit register. These registers are initialized to the following values in hexadecimal, low-order bytes

Process Message in 16-Word Blocks:

We first define four auxiliary functions that each take as input three 32-bit words and produce as output one 32-bit word.

Output:

The message digest produced as output is A, B, C, D. That is, we begin with the low-order byte of A, and end with the high-order byte of D.

Example:

MD5 encoding consists of 128 bits that are represented by a number of 32 symbols hexadecimal, the following examples correspond to ASCII code handled with MD5:

- MD5 ("Generating an MD5 from a text") = 5df9f63916ebf8528697b629022993e8

A small change in the text (change '5' to 'S') produces a completely different output.

- MD5 ("Generating an MDS from a text") = e14a3ff5b5e67ede599cac94358e1028 Other

Security:

Despite being initially considered cryptographically secure, certain investigations have revealed vulnerabilities that make the future use of MD5 questionable. In August 2004, Xiaoyun Wang, Dengguo Feng, Xuejia Lai, and Hongbo Yu announced the discovery of hash collisions for MD5. His attack was consummated in one hour of computation with an IBM P690 cluster

GitHub with example codes.
https://github.com/johnnyzaet08/TareaExtra_DatosII_3_4.git

References

Tutorials Point. (s.f). Cryptography Hash functions. Recuperado de: https://www.tutorialspoint.com/cryptography/cryptography_hash_functions.htm

Rivest, R. L. (April de 1992). The MD5 Message-Digest Algorithm. Obtenido de https://www.ietf.org/rfc/rfc1321.txt