Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X

IJCSMC, Vol. 4, Issue. 3, March 2015, pg.61 – 67

RESEARCH ARTICLE

IMPLEMENTATION OF SECURE HASH ALGORITHM SHA-1 BY LABVIEW

Raaed K. Ibrahim, Ali SH. Hussain, Roula A. Kadhim

[1]Computer Engineering Techniques & Middle Technical University, Iraq

[2]Computer Engineering Techniques & Middle Technical University, Iraq

[3]Computer Engineering Techniques & Middle Technical University, Iraq

[1] raid_khalid2000@yahoo.com, [2] mudarbeen@yahoo.com; [3] roula.aj9@gmail.com

Abstract— This paper presents the implementation of secure hash algorithm using VI LabVIEW environment toolkit. The SHA-1 used in many fields of security systems such as digital signature, tamper detection, password protection and so on. SHA-1 is a very important algorithm for integrity and authentication realization. The proposed algorithm of SHA-1 in this paper implemented by LabVIEW software from entering string, padding, SHA-1 core, and message digest to produce 160 bits hash code for any plaintext.

From the implementation and simulation results of SHA-1 hash function obtained in LabVIEW project show that simplicity in modelling hashing algorithm, generating hash codes in English plaintext, Arabic plaintext, symbols, and numbers.

Keywords— SHA-1 , integrity , hash function ,LabVIEW ,password security

I. INTRODUCTION

This Cryptography is one of the most useful fields in the wireless communication area and personal communication systems, where information security has become more and more important area of interest. Cryptographic algorithms take care of specific information on security requirements such as data integrity, confidentiality and data origin authentication. (Iyer & Mandal, 2013)

A hash function takes a variable sized input message and produces a fixed-sized output. The output is usually referred to as the hash code or the hash value or the message digest (Kak, 2014), hash functions play a significant role in today's cryptographic applications. SHA (Secure Hash Algorithm) is a famous message compress standard used in computer cryptography, it can compress a long message to become a short message abstract (Iyer & Mandal, 2013). In this paper, SHA-1 is implemented using LabVIEW.

SHA-1 is a cryptographic hash function designed by National Security Agency (NSA) and published by National Institute of Standard and Technology (NIST) as a U.S Federal Information Processing Standard (FIPS). SHA stands for Secure Hash Algorithm, the four algorithms for secure hash functions SHA-0,SHA-1 ,SHA-2 and SHA-3 .SHA-1 is very similar to the standard SHA-0 but corrects an error in the original SHA hash specification that led to significant weakness , SHA-1 was first originated in 1995 and currently the most widely used SHA hash function. It is currently used in a wide variety of applications, including TLS, SSL, SSH and PGP. (Wikipedia, 2014).

LabVIEW system design software is at the center of the National Instruments platform. Providing comprehensive tools that is needed to build any measurement or control application in dramatically less time, LabVIEW is the ideal development environment for innovation, discovery, and accelerated results. Combining the power of LabVIEW software with modular, reconfigurable hardware to overcome the ever-increasing complexity involved in delivering measurement and control systems on time and under budget. (Instruments, 2014).

II. SHA-1 ALGORITHM USING LABVIEW

The secure hash algorithm SHA-1 steps implemented by using LabVIEW which has analyzed the LabVIEW environment capabilities for efficient implementation of cryptographic algorithms. The procedure of SHA-1 is illustrated in figure 1.

From figure 1 SHA1 is used to compute a message digest for a message or data file provided as input should be considered as a message or data file is a bit string. The length of the message should not exceed (2^{64})-1 the number of bits can represent in Hex, the message digest size is 160 bits (20 bytes , 40 digit Hex). The message length divided into chunks, each chunk with size of 512 bits. Every chunk processed by identifying five buffers which are A ,B ,C ,D and E . Those buffers pass through functions affected by constants and words (Wt,Kt) to give the final hash code.

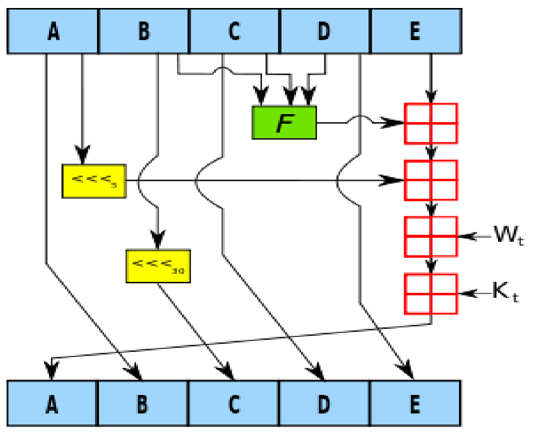


Figure 1: SHA-1 Algorithm (Wikipedia, 2014)

The proposed algorithm is built via LabVIEW where the message will enter to the SHA-1 block and processed to give a hash code , as shown in figure 2.a.

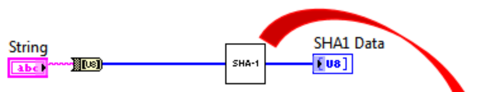


Figure 2.a: LabVIEW Block Diagram

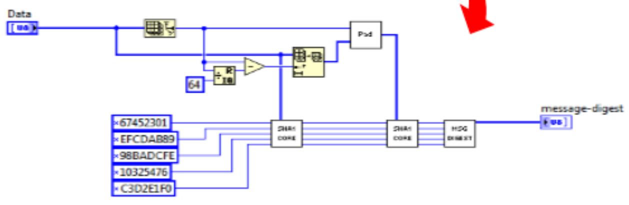


Figure 2.b: SHA-1 block content

The original message or the string is entered to SHA-1 block to produce message digest as shown in figure 2.a. SHA-1 block works as a black box that contains many calculations and constants which represent the SHA-1 specific algorithm. The block diagram shown in figure 2.b is divided into three stages the first stage represent the padding, the second stage is SHA-1 core, and the third is the message digest.

1. Padding The message padding means expanding the length of message within concentration, the purpose of message padding is to make the total length of a padded message congruent to 448 module 512. The number of

padding bits is between 1 and 512. Padding consists of 1 single 1-bit followed by a series of 0-bit. (Chan & Liu, 2007).
Appending Length, a 64-bit binary representation of the original length of the message is appended to the end of the message, this 64 bits to show the original message length before padding extended to the end of message. (Iyer & Mandal, 2013).
The message or data file should be considered to be a bit string, if the number of bits in a message is a multiple of 8, for compactness we can represent the message in hex. The purpose of message padding is to make the total length of a padded message a multiple of 512. SHA-1 sequentially processes blocks of 512 bits when computing the message digest. As a summary, 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. The padded message is then processed by the SHA-1 as n 512-bit blocks. (Eastlake & Jones, 2001).

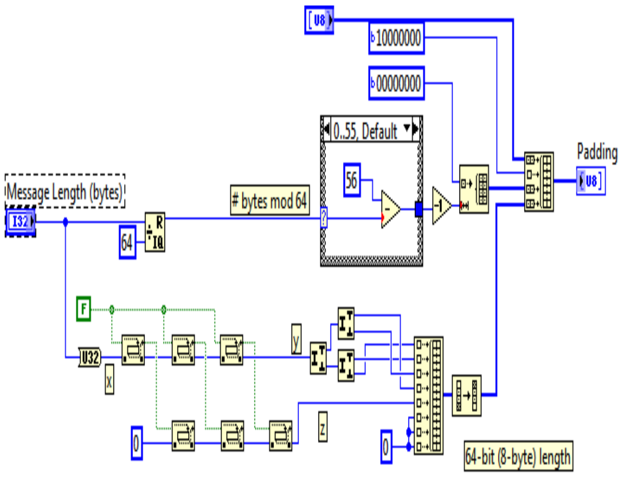


Figure 3: SHA-1 Padding

The padding procedure is divided into two stages, as shown in figure 3 . The first stage when the message enters the loop is converted to bytes (divided by 8), the message enters chunk by chunk which means each 512 bits are processed alone with sequence, but enters as a bytes (64 byte). The second stage is the message shifted left three times and then enters to the last block which has four options, only one released to produce the padded message.

2. SHA-1 Core SHA-1 Core contains the whole system calculations and functions, the input to SHA-1 core are the initial vectors and the padded message.
The initial vectors (IV) are (H0, H1, H2, H3, H4) are initialized as fixed constants, i.e. H0=01234567, H1=89ABCDEF, H2= FECDBA98, H3=76543210, and H4=C3D2E1F0 (Eastlake & Jones, 2001), as shown in figure 4.

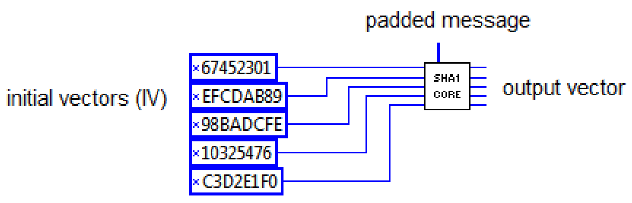


Figure 4: Initial Vector

The next stage in the core is that SHA-1 consists of 4 rounds, each containing 20 iterations (i.e. 80 iterations in total). The algorithm operates on a 128-bit state, divided into four 32-bit words (Patel & Chaudhary, 2012).
The first operation is the allocation of 80 words array for the message schedule to the 80 rounds, as shown in figure 5

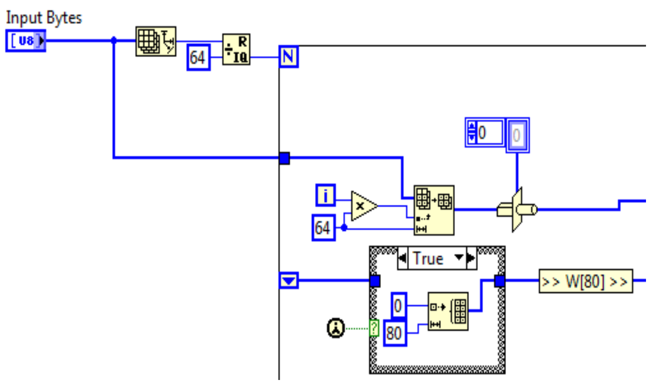


Figure 5: 80 words Array Allocation

The input bytes of figure 5 are the padded message bytes (already 512 bits but enters the loop as bytes). The block diagram built by LabVIEW shows the case structure with Boolean type, in the true case the output gives 80 words with array representation. Else in the false case the function inside the case structure replaced by a single wire which means that the structure works if and only if the first call function works to give a TRUE state. After allocating the 80 words array the procedure comes successively, set the first 16 words to be the 512 bit block split into 16 words, the rest of the words are generated using the logical function:

Word [i3] XOR word [i8] XOR word [i14] XOR word [i16], and then rotated 1 bit to the left, as shown in figure 6.

The LabVIEW block diagram shown in figure 6 shows that the 3rd, 8th, 14th, and 16th words XORed and rotated to left.

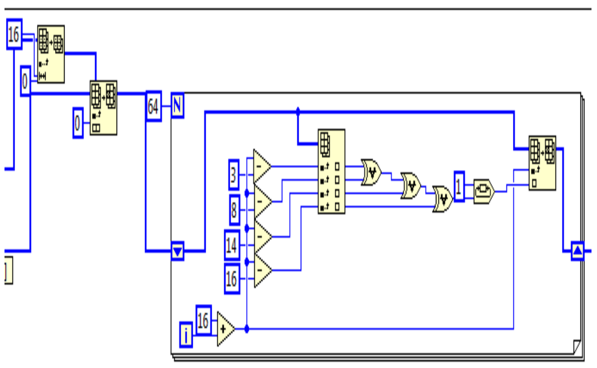


Figure 6: words array iterations

Next the functions and the constants identified, a sequence of logical functions f (0), f (1)... f (79) is used in SHA-1. Each f (t), 0 <= t <= 79, operates on three 32-bit words B, C, D and produces a 32-bit word as output. f(t;B,C,D) is defined as follows:

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). (Eastlake & Jones, 2001) (Lawrence C. Washington), these operations implemented in LabVIEW blocks as shown in figure 7.

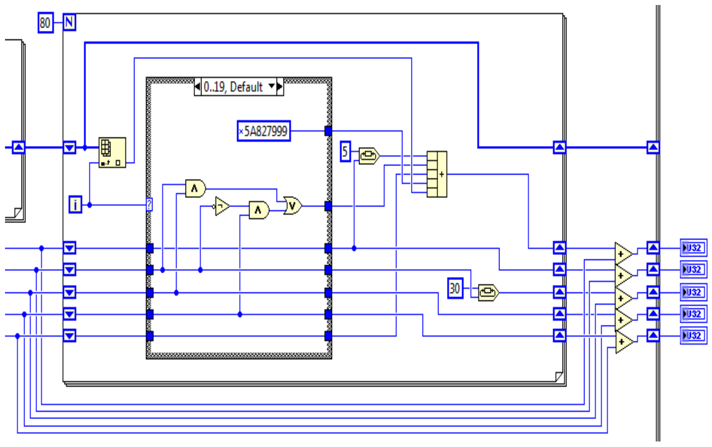


Figure 7: SHA-1 Calculations

A sequence of constant words K(0), K(1), ... , K(79) is used in the SHA-1. In hex these are given by:

K(t) = 5A827999 (0 <= t <= 19)
K(t) = 6ED9EBA1 (20 <= t <= 39)
K(t) = 8F1BBCDC (40 <= t <= 59)
K(t) = CA62C1D6 (60 <= t <= 79). (Eastlake & Jones, 2001) (Lawrence C. Washington) (Yang, 2014).

From figure 7, the 80 rounds of hash algorithm started, inside these rounds there is a case structure which has four iterations, for each iteration the logical operation converted and also the constant changed according to t value, as shown in figure 8.

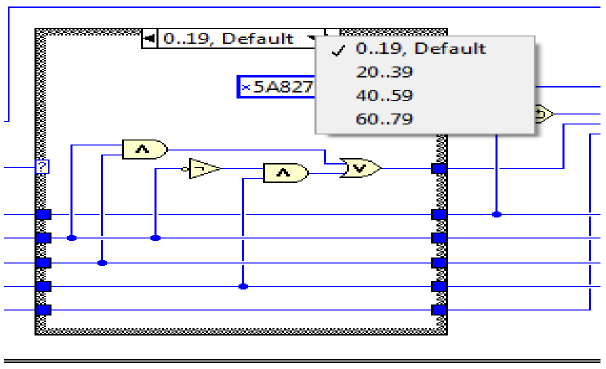


Figure 8.a: SHA-1 functions with LabVIEW Case Structure

Figure 8.a shows that as default the first function is:

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

with constant value K(t) = 5A827999 for period (0 <= t <= 19), the next case of function and constant changed by the second iteration from 19 to 20 based on t value, as shown in figure 8.b. The same procedure to the next iterations.

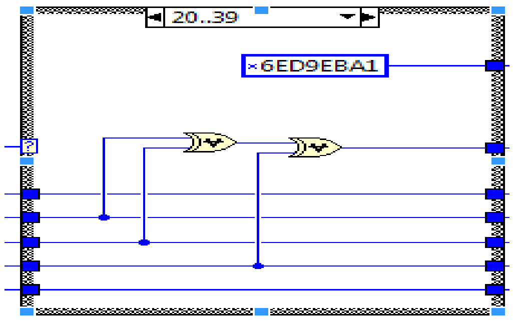


Figure 8.b: the second round in the case structure

3. Massage Digest The last stage of the SHA-1 core is the message digest which is the last calculation to give the final code which is the hash, as shown in figure 2.b the message digest input is the output vector. The message digest is computed using the padded message. The message digest (VI) block diagram is shown in figure 9.

The computation is described using two buffers, each consisting of five 32-bit words, and a sequence of eighty 32-bit words. The words of the first 5-word buffer are labeled A,B,C,D, and E. 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).A single word buffer TEMP is also employed. (Eastlake & Jones, 2001)

To generate the message digest, the 16-word blocks M(1), M(2),...,M(n) are processed in order. The processing of each M(i) involves 80 step.

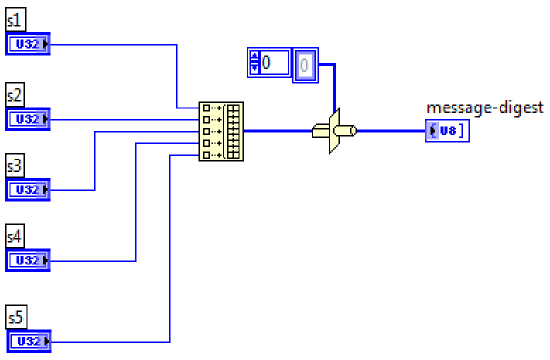


Figure 9: Message Digest

Now M(1), M(2), ... , M(n) are processed. To process M(i) 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) ,show figure 7.; B = A; A = TEMP

Let H0 = H0 + A, H1 = H1 + B, H2 = H2 + C, H3 = H3 + D, H4 = H4+ E

After processing M(n), the message digest is the 160-bit string represented by the 5 words : H0 H1 H2 H3 H4. (Eastlake & Jones, 2001).

From figure 9 it is distinct that the inputs are the output of SHA-1 core as illustrated in figure 7 ,these inputs will be processed by a function and changes its type to Hex to give the last representation of the code which is the hash code (digest) with 160 bits length.

III.RESULTS

From the implementation of cryptography hash function in (VI) LabVIEW 2012 and test the execution of all steps in SHA-1 algorithm in different types of plaintext such as English , Arabic ,symbols ,and numbers to produce fixed 160 bits hash code.

The testing figures of SHA-1 160 bits hash function as shown in figures(11.a,b,c).

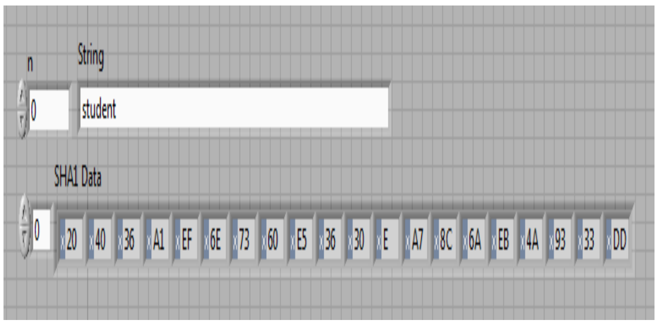
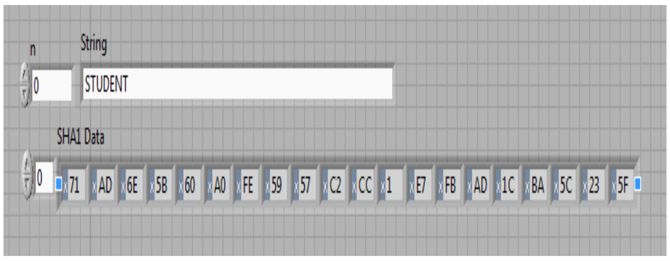

n
0
String
student
SHA1 Data
0
20 40 36 A1 EF 6E 73 60 E5 36 30 E A7 8C 6A EB 4A 93 33 DD

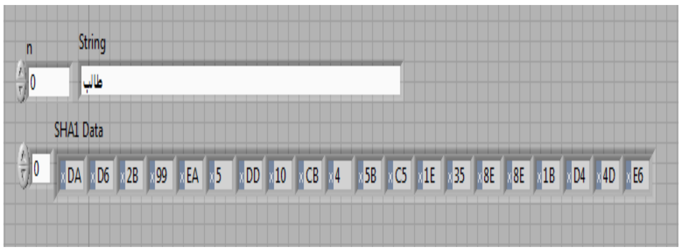
Figure 11.a : Testing the system (English lowercase state)



```
n       String
0       STUDENT

SHA1 Data
0   71 AD 6E 5B 60 A0 FE 59 57 C2 CC 1 E7 FB AD 1C BA 5C 23 5F
```

Figure 11.b: Testing the system (English uppercase state)



```
n       String
0       طالب

SHA1 Data
0   DA D6 2B 99 EA 5 DD 10 CB 4 5B C5 1E 35 8E 8E 1B D4 4D E6
```

Figure 11.c: Testing the system (Arabic state)

The text entered to the system is "student" which is 7 bytes in size, after processing via SHA-1 algorithm the message digest will be : 204036A1EF6E7360E536300EA78C6AEB4A9333DD.

The most important property of the system is that any change in the text, the message digest changes, also if there is any letter exchanging from uppercase to lowercase and vice versa, this is called avalanche.

IV. CONCLUSIONS

From the results of SHA-1 built via (VI) LabVIEW are concluded the following points:

1. All the previous implementation of SHA-1 used either high level software or middle level software which are very complicated methods to obtain the idea of SHA-1 hash function, but by using LabVIEW it is very easy software to learn the idea of SHA-1 algorithm.
2. The process speed of (VI) LabVIEW to implement SHA-1 algorithm is very acceptable with respect to other software tool and very easy method to compute the run time.
3. From the proposed implementation of SHA-1 in LabVIEW it is very easy to control and change the coefficient parameters in SHA-1 algorithm due to the simplicity and flexibility of (VI) LabVIEW tasks, so it is very easy to compute the hash codes of English and other languages in our proposed system.

References

[1] X. &. L. G. Chan, Discussion of One Improved Hash Algorithm Based on MD5 and SHA1, San Francisco, USA: World Congress on Engineering and Computer Science (WCECS), 2007.

[2] D. &. J. P. Eastlake, “Network Working Group, 3rd Motorola , Cisco System,” US Secure Hash Algorithm 1 (SHA1). , Retrieved from RFC 3174 - US Secure Hash Algorithm 1 (SHA1), 2001.

[3] N. Instruments, “http://www.ni.com/labview/why,” National Instruments, 18 sep. 2014. [Online]. Available: http://www.ni.com. [Accessed 10 dec. 2014].

[4] N. &. M. S. Iyer, “Implementation of Secure Hash Algorithm-1 using FPGA,” *International Journal of Information and Computation Technology. ,* vol. III, no. 3, pp. 50-60, 2013.

[5] A. Kak, “Hashing for Message Authentication,” *Computer and Network Security,* vol. 5, no. 4, pp. 332-339, 2014.

[6] L. C. W. (n.d.)., Introduction to Cryptography, Washington : Wireless Information Network Laboratory., 2010.

[7] R. &. C. N. Patel, “Analyzing Digital Signature Robustness with Message Digest Algorithms,” *Computer Applications on Communication Security.,* vol. 6, no. 3, pp. 8875-8887, 2012.

[8] t. f. Wikipedia, “SHA-1,” Wikipedia, 19 November 2014. [Online]. Available: http://en.wikipedia.org/wiki/SHA-1. [Accessed 3 jan. 2015].

[9] D. H. Yang, “Cryptography Tutorials - Herong's Tutorial Examples,” Retrieved from SHA1 Mesasge Digest Algorithm, 10 sep. 2014. [Online]. Available: http://www.herongyang.com/Cryptography/SHA1-Message-Digest-Algorithm-Overview.html. [Accessed 10 jan. 2015].