ISBN 978-93-84422-22-6
Proceedings of 2015 International Conference on Image Processing, Production and Computer Science (ICIPCS'2015)
Istanbul (Turkey), June 3-4, 2015 pp. 112-119

Implementation of Secure Hash Algorithm Sha-2 256 by using Labview

Author Roula AJ. Kadhim[1], Raaed K. Ibrahim[2], Ali SH. Alkhalid[3]

[1] College of Elec. & Electronic engineering Techniques

[2] College of Elec. & Electronic engineering Techniques

[3]College of Elec. & Electronic engineering Techniques

***Abstract*:** *This paper presents the implementation of secure hash algorithm SHA-2 using VI LabVIEW environment toolkit. The SHA-2 used in many fields of security systems such as digital signature, tamper detection, password protection and so on. SHA-2 is a very important algorithm for integrity and authentication realization. The proposed algorithm of SHA-2 in this paper implemented by LabVIEW software from entering string, padding, SHA-2 core, and message digest to produce 256 bits hash code for any plaintext.*
From the implementation and simulation results of SHA-2 hash function obtained in LabVIEW project show that simplicity in modelling hash algorithm, generating hash codes in English plaintext, Arabic plaintext, symbols, and numbers.

***Keywords**: SHA-2, integrity, authentication, hash function, LabVIEW.*

1. Introduction

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 and data origin authentication. [1] 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, hash functions play a significant role in today's cryptographic applications. [2] [3] [4]. Secure Hash Algorithm (SHA) is the most widely used Hash Function in the world. It was developed by the National Institute of Standards and Technology (NIST) in the United States and first published in 1993. This version (SHA-0) was found to have a serious security flaw, though NIST never published the details of this, and was replaced in 1995 with SHA-1. In recent years, SHA-1 has been found to have weaknesses, meaning a collision may be found. It was suggested in that with a $1 million budget, a message could be broken in 25 days. Due to this, NIST developed SHA-2, which has a larger output (256 or 512-bit over the 160-bit in SHA-1) and differences within the message computation. Due to the Preimage resistance of SHA, the method of operation can be made public. Would-be attackers gain no benefit from knowledge of the SHA Algorithm, as no key is used to create the message digest. [5].

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. [6].

2. SHA-2 Algorithm Using LabVIEW

The secure hash algorithm SHA-2 steps implemented by using LabVIEW which has analyzed the LabVIEW environment capabilities for efficient implementation of cryptographic algorithms. The procedure of SHA-2 is illustrated in figure 1.

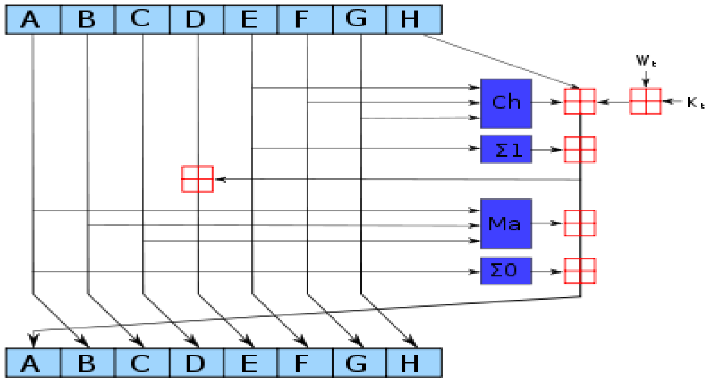


Fig. 1: SHA-2 Algorithm

From figure 1 SHA-2 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 256 bits (32 bytes, 64 digit Hex). The message length divided into chunks, each chunk with size of 512 bits the same as the previous secure hash algorithm SHA-1. Every chunk processed by identifying eight buffers which are A ,B ,C ,D ,E ,F ,G and H . Those buffers pass through functions affected by constants and words (Wt, Kt) to give the final hash code. [7] [8]

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

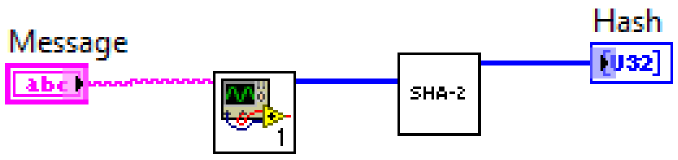


Fig. 2: SHA-2 Block Diagram

In LabVIEW the SHA-2 block which shown in figure 2 is include the whole system that implement the calculation of the algorithm, inside it there are two blocks as shown in figure 3.

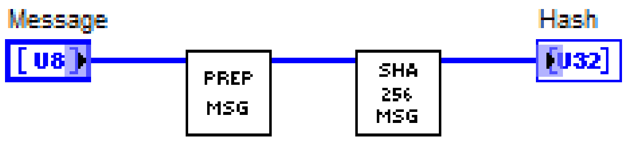


Fig. 3: SHA 256 Block diagram

The original message or the string is entered to SHA-2 block to produce message digest as shown in figure 2. SHA-2 block works as a black box that contains many calculations and constants which represent the SHA-2 specific algorithm. From figure 3 notice that the message passes across two main stages, the first stage represent the preparing of the original message to make it suitable to be processed via the next stage which is the SHA-2 256 MSG.

2. Preparing the Message

The pre-processing consists of padding the message purposes to ensure that the padded message is a multiple of 512 bits. [2]. The padding module appends the bit "1" to the end of the message, followed by multiple-zero bits. Then, 64 bit showing length of the original message is added after. [9] [10]

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. [11] [12] [13]. The first stage which is the preparing of the message classify into two branches as shown in figure 4.

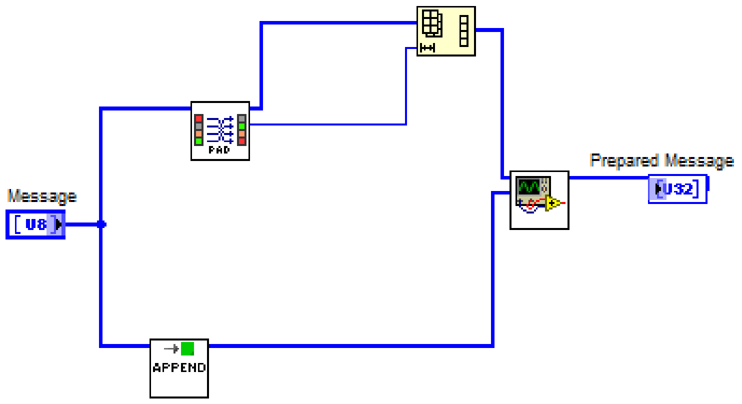


Fig. 4: Preparing of the message

From figure 4 notice that the first branch include the padding which means pad the message with one 1 and a series of 0's until the number of bits modulo 512 is equal to 448, the second branch responsible for appending the length of the message with 64 unsigned integer which is show the size of the original message before padding. The output of this block is the prepared message which is the padded one which is ready to process by SHA-2.

3. SHA-2 256 Message

The second stage of figure 3 is SHA-2 256 MSG; inside this block there is many calculations to give at the end the unique hash code, as shown in figure 5.

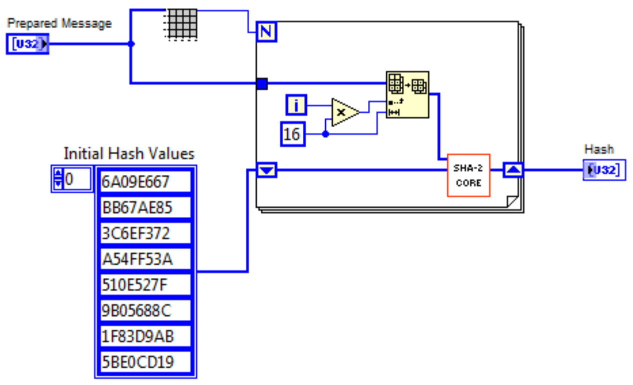


Fig. 5: SHA-2 256 Block diagram

SHA-2 Core contains the whole system calculations and functions, the input to SHA-2 core are the initial vectors and the padded message. The initial vectors (IV): (H0, H1, H2, H3, H4, H5, H6, and H7) are initialized as fixed constants each one is 32 bits word, which are: H0 = 6A09E667, H1= BB67AE85, H2= 3C6EF372, H3= A54FF53A, H4= 510E527F, H5 = 9B05688C, H6 = 1F83D9AB, and H7 = 5BE0CD19 as shown in figure 5. [10]. [13] [14] [1]. [15].

4. SHA-2 Core:

The Message is processed in 512-bit blocks sequentially, just like SHA-1, which means that if the original message length less than 448, the message prepared by padding to be congruent to 448 mod 512. Otherwise , if the message more than 512 then the message is divided into chunks or blocks, each with size of 512 bit to be processed sequentially by SHA-2. The difference in SHA-2 is the message digest or the hash code is 256 bits instead of 160 bits in SHA-1. The second different is there are 64 rounds instead of 80 round in SHA-1. The third different is shift right operation which used in SHA-2 in addition to rotate right, in SHA-1 there is no shift. Show figure 6. [5] [8].

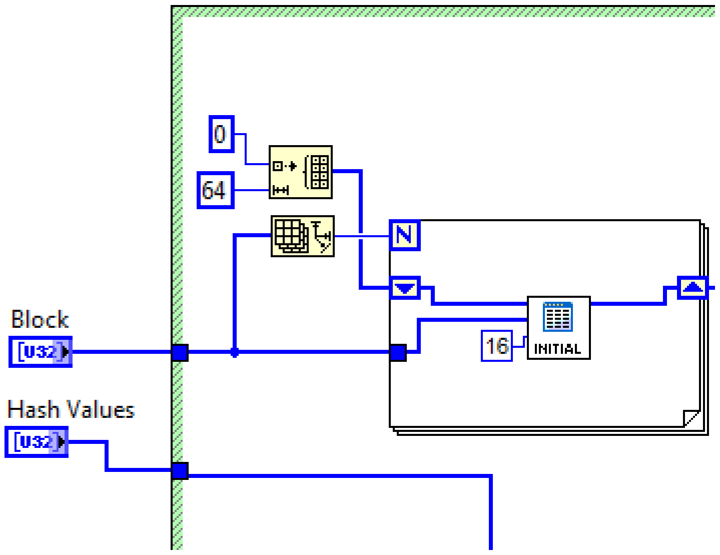


Fig. 6: Creation of a 64-entry message schedule array

From figure 6, for each chunk create a 64-entry message schedule array w [0...63] of 32-bit words, copy chunk into first 16 words w [0...15] of the message schedule array, the initial vectors IV or the hash values entered to the system. Extend the first 16 words into the remaining 48 words w [16...63] of the message schedule array. [13] [2]

For i from 16 to 64

s0:= (w [i-15] ROT 7) xor (w [i-15] ROT 18) xor (w [i-15] SHR 3)

s1:= (w [i-2] ROT 17) xor (w [i-2] ROT 19) xor (w [i-2] SHR 10)

W[i]:= w [i-16] + s0 + w [i-7] + s1. Show figure 7. [7] [14] [1] [15]

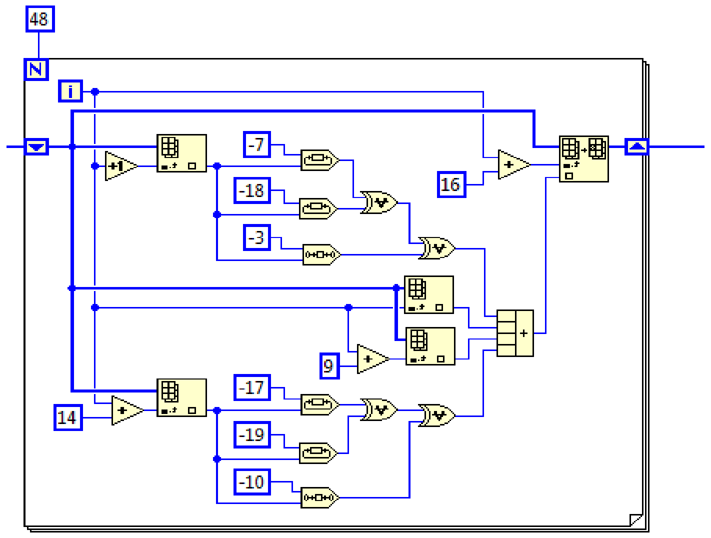


Fig. 7: words array iterations

The next step is Initialize working variables to current hash value:

A: = H0, B: = H1, C: = H2, D: = H3, E: = H4, F: = H5, G: = H6, H: = H7. [14]

Then the Compression function main loop:

For i from 0 to 63

S1:= (E ROT 6) xor (E ROT 11) xor (E ROT 25)

Ch: = (E AND F) xor ((NOT E) AND G)

temp1:= H + S1 + Ch + k[i] + w[i]

S0:= (A ROT 2) xor (A ROT 13) xor (A ROT 22)

Maj: = (A AND B) xor (A AND C) xor (B AND C)

temp2:= S0 + Maj, show figure 1, 8. [7] [2] [1] [15].

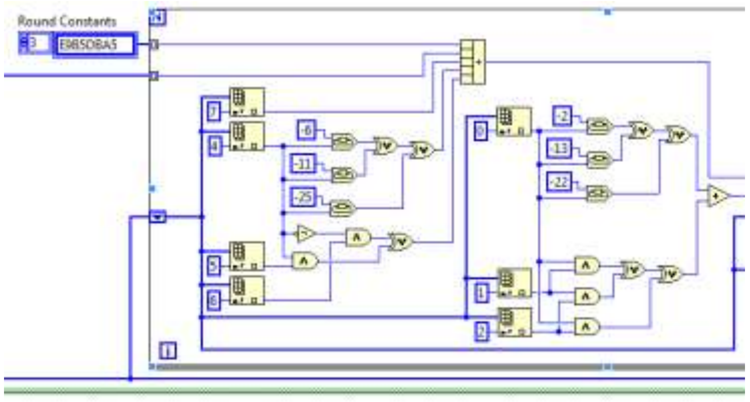


Fig. 8: SHA-2 calculations

Each round of the 64 rounds has a different constant which is continuously changed during execution, within these rounds initialize two temporary registers which are temp1, temp2. After that the values of the variables exchanged as H: = G, G: = F, F: = E, E: = D+ temp1, D: = C, C: = B, B: = A, A: = temp1 + temp2. Show figure 1. [7].

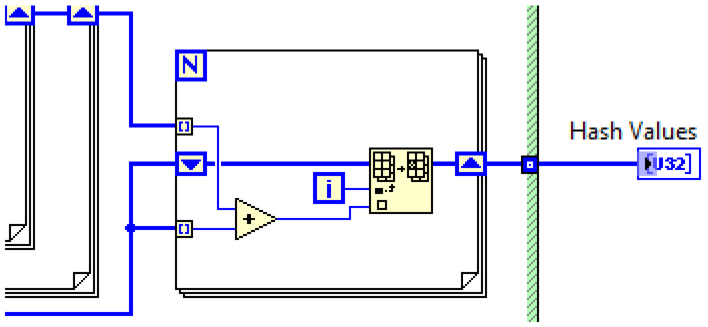


Fig. 9: Finalize hash values

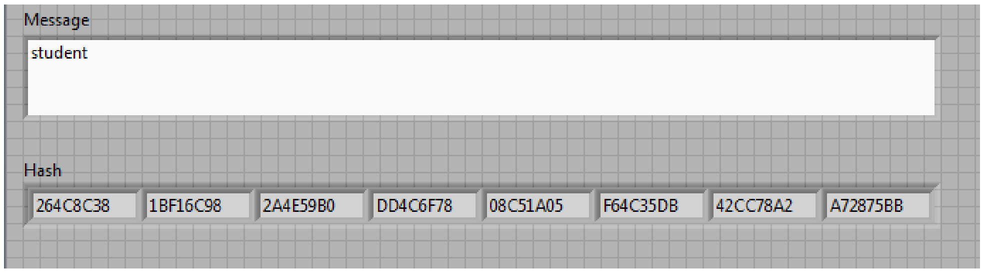
After the 64 rounds of SHA-2 core the presentation of the hash code appears except the final stage which is by adding the compressed chunk to the current hash value (updating the hash vectors).

H0:= H0 + A, H1:= H1 + B, H2:= H2 + C, H3:= H3 + D, H4:= H4 + E, H5:= H5 + F, H6:= H6 + G, H7:= H7 + H. [7]

Finally, to produce the hash code organize these vectors in big endian, so the hash code is: H0 append H1 append H2 append H3 append H4 append H5 append H6 append H7 to give the 256 bits hash code. [2] [1]. Show figure 9.

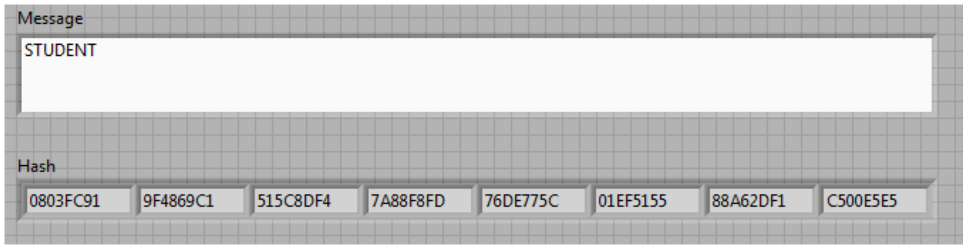
5. Results and Conclusions

From the implementation of cryptography hash function in (VI) LabVIEW 2012 and test the execution of all steps in SHA-2 algorithm in different types of plaintext such as English , Arabic ,symbols ,and numbers to produce fixed 256 bits hash code. The testing figures of SHA-2 256 bits hash function as shown in figure 10 (A, B, C).


Message
student

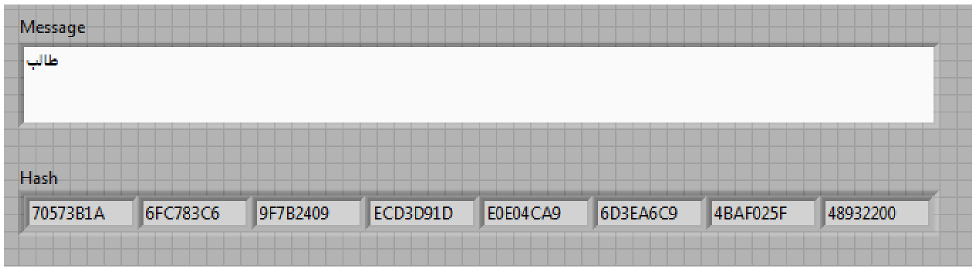
Hash
264C8C38 1BF16C98 2A4E59B0 DD4C6F78 08C51A05 F64C35DB 42CC78A2 A72875BB

Fig. 10.A: testing the system (English lowercase)


Message
STUDENT

Hash
0803FC91 9F4869C1 515C8DF4 7A88F8FD 76DE775C 01EF5155 88A62DF1 C500E5E5

Fig 10.B: testing the system (English uppercase)


Message
طالب

Hash
70573B1A 6FC783C6 9F7B2409 ECD3D91D E0E04CA9 6D3EA6C9 4BAF025F 48932200

Fig 10.C: testing the system (Arabic)

The text entered to the system is "student" which is 7 bytes in size, after processing via SHA-2 algorithm the message digest which is unique will be:

264C8C38 1BF16C98 2A4E59B0 DD4C6F78 08C51A05 F64C35DB 42CC78A2 A72875BB.

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. From the results of SHA-2 built via (VI) LabVIEW are concluded the following points:

1. All the previous implementation of SHA-2 used either high level software or middle level software which is very complicated methods to obtain the idea of SHA-2 hash function, but by using LabVIEW it is very easy software to learn the idea of SHA-2 algorithm.
2. The process speed of (VI) LabVIEW to implement SHA-2 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-2 in LabVIEW it is very easy to control and change the coefficient parameters in SHA-2 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.
4. By comparing between SHA-1 and SHA-2 implementation via LabVIEW notice that the hash code to the same word in SHA-1 is 204036A1EF6E7360E536300EA78C6AEB4A9333DD which is 160 bits but in SHA-2 is 264C8C38 1BF16C98 2A4E59B0 DD4C6F78 08C51A05 F64C35DB 42CC78A2 A72875BB which is 256. The increasing of number of bits in the hash code make it more difficult to brake and it is a hint that the system is strong and less collisions.
5. For cryptographic hash function, the following property is required, Preimage resistance: it is computationally infeasible to find any input which hashes to any pre-specified output which is obtained by the implementing SHA-2 algorithm which is unbreakable till now.

6. References

[1] SECURE HASH STANDARD, Federal Information Processing Standards Publication 180-2, 2002.

[2] R. V. Mankar and S. I. Nipanikar, "C Implementation of SHA-256 Algorithm," International Journal of Emerging Technology and Advanced Engineering, pp. 167-170, 2013.

[3] C. Knopf, Cryptographic Hash Functions, 2007.

[4] J. Rompel, "One-Way Functions are Necessary and Sufficient for Secure Signatures," in Laboratory for Computer Science Massachusetts Institute of Technology, Cambridge,, 1990.
http://dx.doi.org/10.1145/100216.100269

[5] J. Docherty and A. Koelmans, Hardware Implementation of SHA-1 and SHA-2 Hash Functions, Newcastle: Microelectronics System Design Research Group ,School of Electrical, Electronic and Computer Engineering, 2011.
http://dx.doi.org/10.1109/iscas.2011.5937967

[6] N. Instruments, "National Instruments," 2014. [Online]. Available: http://www.ni.com/labview/why.

[7] M. Juliato, . C. Gebotys and R. Elbaz, "Efficient Fault Tolerant SHA-2 Hash Functions for Space Applications," 2009.

[8] M. K. R. Danda, DESIGN AND ANALYSIS OF HASH FUNCTIONS, 2007.

[9] H. A. Tuan, K. Yamazaki and S. Oyanagi, "THREE-STAGE PIPELINE IMPLEMENTATION FOR SHA2 USING DATA FORWARDING," pp. 29-34, 2008.

[10] A. L. Selvakumar and C. Ganandhas , "The Evaluation Report of SHA-256 Crypt Analysis Hash Function," International Conference on Communication Software and Networks, pp. 588-592, 2009.

[11] R. K. Ibrahim, A. S. Hussain and R. A. Kadhim, "IMPLEMENTATION OF SECURE HASH ALGORITHM SHA-1 BY LABVIEW," International Journal of Computer Science and Mobile Computing, pp. 61-67, 2015.

[12] X. Chan and G. Liu, "Discussion of One Improved Hash Algorithm Based on MD5 and SHA1," World Cngress on Engineering and Computer Science (WCECS), San Francisco, USA, 2007.

[13] R. P. McEvoy , F. M. Crowe , C. C. Murph and W. P. Marnane, "Optimisation of the SHA-2 Family of Hash Functions on FPGAs," Irish Research Council for Science Engineering and Technology (IRCSET)., 2005.

[14] I. Nikoli´ and A. Biryukov, "Collisions for Step-Reduced SHA-256," International Association for Cryptologic Research, 2008.

http://dx.doi.org/10.1007/978-3-540-71039-4_1

[15] C. H. Romine, FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION, Patrick Gallagher, Under Secretary for Standards and Technology and Director, 2012.