Available online at http://www.journalcra.com

International Journal of Current Research
Vol. 9, Issue, 10, pp.58607-58611, October, 2017

INTERNATIONAL JOURNAL OF CURRENT RESEARCH

ISSN: 0975-833X

RESEARCH ARTICLE

FPGA IMPLEMENTATION OF ENHANCED SHA-192 ALGORITHM

*Shiva Nagender Rao, L., Sai Rishishwar Reddy, Y. and Manoj Yasaswi, V.

UG Scholar, Dept of ECE, SNIST, Hyderabad

ARTICLE INFO

Article History:
Received 15th July, 2017
Received in revised form
07th August, 2017
Accepted 24th September, 2017
Published online 17th October, 2017

Key words:

SHA, Cryptology, TLS, SSL.

ABSTRACT

Hash functions were introduced in Cryptology as a tool to protect the integrity of information. Secure Hash Algorithm-1 (SHA-1) and Message Digest-5 (MD-5) are among the most commonly used hash function message digest algorithms. Scientists have found collision attacks on SHA-1, MD-5 hash functions so the natural response to overcome this threat was assessing the weak points of these protocols that actually depend on collision resistance for their security. So to increase the security, modified SHA-192 is introduced in this paper having a message digest of length 192 bits with larger bit difference. To generate larger bit difference, best properties of MD-5andSHA-1 are combined. So the new solution will be no longer vulnerable to the collision attacks.SHA-192 currently used in security applications and protocols applications including Transport Layer Security (TLS), Secure Socket Layer (SSL), Internet Protocol Security (IPSec) and as Digital Signatures. This technique is designed by using Verilog HDL with Xilinx ISE Design suite 12.4 version tool. The designs implemented in Xilinx SPARTAN 3E XC3S500EFG320 FPGA board.

Copyright©2017, Shiva Nagender Rao et al. *This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.*

Citation: Shiva Nagender Rao, L., Sai Rishishwar Reddy, Y. and Manoj Yasaswi, V. 2017. "FPGA implementation of enhanced SHA-192 algorithm", *International Journal of Current Research*, 9, (10), 58607-58611.

INTRODUCTION

Mobile computers have become an essential tool for many of our daily activities. As we turn increasingly to portable devices for our professional and personal needs, we are left with little choice but to entrust sensitive information to digital media. Since these devices are susceptible to physical theft, we must rely on cryptographic algorithms to ensure the confidentiality and integrity of our personal data. Thus, the security of this data rests on the inability of an attacker to guess one important piece of information. In this age of universal electronic connectivity, of viruses and hackers, of electronic eavesdropping and electronic fraud, there is indeed no time at which security does not matter. Two trends together made me to choose this topic of vital interest. First, the explosive growth in computer systems and their interconnections via networks has increased the dependence of both organizations and individuals on the information stored and communicated using these systems. This, in turn, has led to heightened awareness of the need to protect data and resources from disclosure, to guarantee the authenticity of data and messages, and to protect systems from network based attacks. Second, the disciplines of cryptography and network security have matured, leading to the development of practical, readily available applications to enforce network security.

Corresponding author: Shiva Nagender Rao,
UG Scholar, Dept of ECE, SNIST, Hyderabad, India.

This security is achieved by using cryptographic algorithms such asymmetric, asymmetric and hash algorithms like MD4 (Message Digest 4), MD5 (Message Digest 5) and SHA (Secure Hash Algorithm) series.

Enhanced sha-192 Algorithm

The most widely used hash function is the Secure Hash Algorithm (SHA) because virtually every other widely used hash function had been found to have substantial cryptanalytic weaknesses, SHA was more or less the last remaining standardized hash algorithm by 2005. SHA was developed by the National Institute of Standards and Technology (NIST) and published as a federal information processing standard (FIPS 180) in 1993. When weaknesses were discovered in SHA, now known as SHA-0, a revised version was issued as FIPS 180-1 in 1995 and is referred to as SHA-1. SHA is based on the hash function MD4, and its design closely models MD4. SHA-1 is also specified in RFC 3174, which essentially duplicates the material in FIPS 180-1 but adds a C code implementation. SHA-1 produces a hash value of 160 bits. In 2002, NIST produced a revised version of the standard, FIPS 180-2, that defined three new versions of SHA, with hash value lengths of 256, 384, and 512 bits, known as SHA-256, SHA-384, and SHA-512, respectively. Collectively, these hash algorithms are known as SHA-2. These new versions have the same underlying structure and use the same types of modular arithmetic and logical binary operations as SHA-1. A revised document was issued as FIP PUB 180-3 in 2008, which added a 224-bit version. SHA-2 is also specified in RFC 4634, which essentially duplicates the material in FIPS 180-3 but adds a C code implementation. The weakness in SHA family is that two different inputs will produce the same output. There is a need to have a good diffusion so that the output in each round will be spread out and not to be equal with the same output in the next coming stages. For a hash function finding a message that corresponds to a given message digest can always be done using a brute force search in 2L evaluations, where L is the number of bits in the message digest. This is called a pre image attack and may or may not be practical in a particular computing environment. The second criterion, finding two different messages that produce the same message digest, known as a collision, requires on average only 2L/2evaluations using a birthday attack. For the latter reason the strength of a hash function is usually compared to a symmetric cipher of half the message digest length. Thus SHA-1 was originally thought to have 80-bit strength but it has been identified that security flaws were identified in SHA-160 indicating that stronger hash function would be desirable. Hence the modifications are proposed enhance the security of the existing algorithm. So to increase the security, modified SHA-192 has been explained in this chapter having a message digest of length 192 bits with larger bit difference. To generate larger bit difference, best properties of MD-5 and SHA-1 are combined. The proposed algorithm will be no longer vulnerable to the collision attacks.

Block Diagram of SHA-192 Algorithm

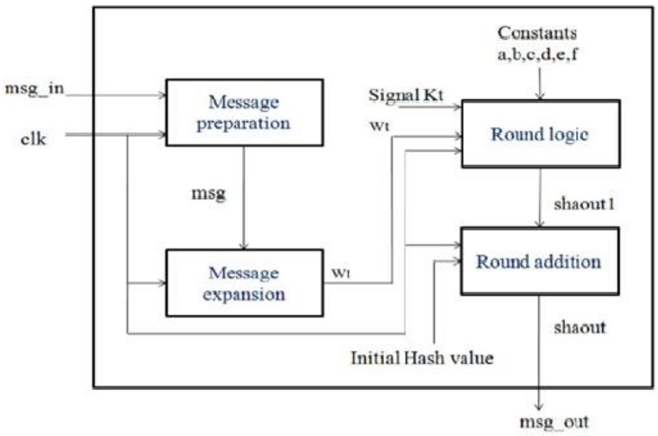


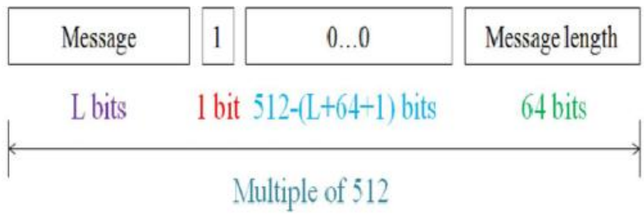
Figure 1. SHA-192 Algorithm consists of four modules which are

- Message preparation.
- Message expansion.
- Round logic and
- Round addition

Firstly, the message input (‘msg_in’) is given to message preparation module which converts it into ‘msg’ according to the principle which will be explained in chapter 3.2. This output is given as input to the message expansion module which will split the given input based on the expansion principle explained in chapter 3.3. Now this message expansion output ‘Wt’ produces the output ‘shaout1’ from the round logic module with the consideration of constants ‘a’, ‘b’, ‘c’, ‘d’, ‘e’ and ‘f’. Finally, the initial hash value is added to ‘shaout1’ to obtain the output ‘shaout’ which is the required message output (‘msg_out’) as explained in chapter 3.5.

Message Preparation

Message preparation includes Padding and appending of the message (‘msg_in’) which is shown in figure 3.2. The purpose of padding is to ensure that the padded message is multiple of 512 bits which is called as a frame. If the length of the message M, is L bits it is append the bit 1 to the end of the message followed by k zero bits, where k is smallest, non-negative solution to the equation L+1+k = 448 mod512. Now, append the 64 bits block that is equal to the number L written in binary. The maximum number of input bits to this block is 2^{64}-1 bits.



For example, let the message ‘M’ be $(01100010\ 11001010\ 1001)_2$ which is of the length L = 20-bits. The value of M is followed by 1-bit ‘1’ and 427 zeroes i.e, ‘512-(20+64+1)’ 0’s. Remaining 64-bits of LSB is the message length represented in 64-bit format of ‘L’ i.e, 20 $((00000000\ 00000000\ 00000000\ 00000000\ 00000000\ 00000000\ 00000000\ 00010100)_2)$. The final output of message preparation in 512-bit format is:

(01100010 11001010 10011000 00000000

Message Expansion

The output of message preparation block is 512 bits for every 447 bits or less. If the length of message exceeds 447 bits, the message will be expanded into 512 bits for every 447 bits by message preparation principle as explained in the chapter 3.2. This module divides the input message into blocks, each of length 512 bits, i.e. cut M into sequence of 512 bit blocks M[1], M[2]…..M[N]. Then the current 512 bit block is divided into 16 sub blocks each consisting of 32 bits. Each of Mi parsed into thirty-two 16 bits words Mi[0], Mi[1]…....Mi [32] which is labeled as Wt as shown in Figure 3.

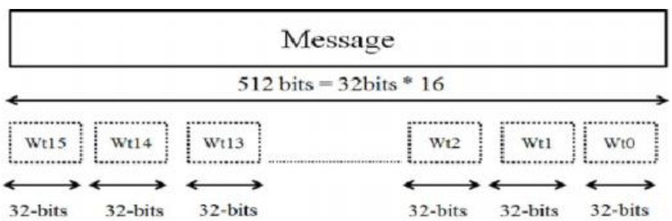


Figure 3. Representation of Message Expansion

For example, let the message 'M' of 512-bits be "(01100010 11001010 10011000 00000000 00010100)$_2$". The output of message expansion Wt0, Wt1, Wt2, ….., Wt13, Wt14 and Wt15 each of 32-bits is as follows:

Wt0 = 00000000 00000000 00000000 00010100,
Wt1 = 00000000 00000000 00000000 00000000,
Wt2 = 00000000 00000000 00000000 00000000,
Wt3 = 00000000 00000000 00000000 00000000,
Wt4 =00000000 00000000 00000000 00000000
Wt5 = 00000000 00000000 00000000 00000000,
Wt6 = 00000000 00000000 00000000 00000000,
Wt7 = 00000000 00000000 00000000 00000000,
Wt8 = 00000000 00000000 00000000 00000000,
Wt9 = 00000000 00000000 00000000 00000000,
Wt10 = 00000000 00000000 00000000 00000000,
Wt11 = 00000000 00000000 00000000 00000000,
Wt12 = 00000000 00000000 00000000 00000000,
Wt13 = 00000000 00000000 00000000 00000000,
Wt14 = 00000000 00000000 00000000 00000000 and
Wt15 = 01100010 11001010 10011000 00000000.

Round Logic

Round Logic is the main functional element of proposed algorithm. Here, the rounded logic output is obtained by performing the logic which is explained later in this chapter in the sections 3.4.1 and 3.4.2.The Round Logic architecture for SHA-192 and Enhanced SHA-192 algorithms shown in the Figures 3.4 and 3.5 which looks similar but the difference is that Enhanced SHA-192 has two stages, while SHA-192 has single stage. Round Logic is the main functional element of proposed algorithm. Here, the rounded logic output is obtained by performing the logic which is explained later in this chapter in the sections 3.4.1 and 3.4.2.The Round Logic architecture for SHA-192 and Enhanced SHA-192 algorithms shown in the figures 3.4 and 3.5 which looks similar but the difference is that Enhanced SHA-192 has two stages, while SHA-192 has single stage.

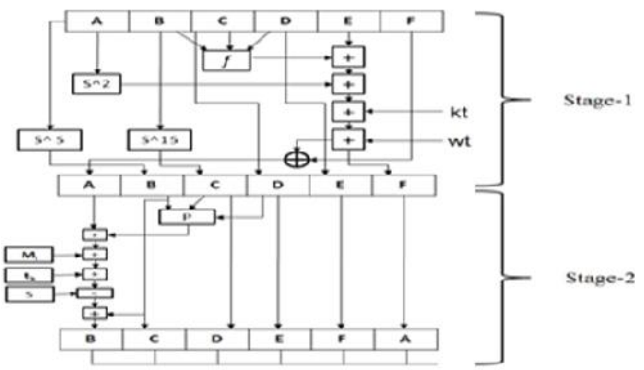


Figure 4. Round Logic Architecture for SHA-192

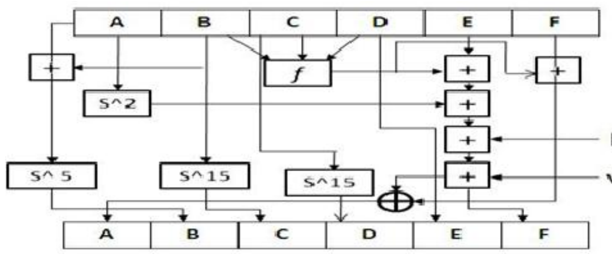


Figure 5. Round Logic Architecture for Enhanced SHA-192

The two steps that constitute the Round Logic are as follows:

- Initialize Chaining variables and
- Processing of Message block

Initialize Chaining Variables

In this step, six chaining variables A through F are initialized. Before the hash function begins, the initial hash value H0 must be set. The hash is 192 bits used to hold the intermediate and final results the hash can be represented as six 32 bit words registers A, B, C, D, E, F.

These values are given as:

A=67452301,
B=EFCDAB8,
C=98BADCF,
D=10325476,
E=C3D2E1F0 and
F=40385172

Processing of Message Block

After initializing the variables, the actual algorithm begins. Each message block is processed in order using following steps:

- Copy the chaining variables A-F into variables a-f. For i=1 to N prepare the message schedule.
- Divide the current 512 bit block into 16 sub blocks each consisting of 32 bits. Each of Mi parsed into thirty-two 16 bits words Mi[0], Mi[1]..Mi [32]. These blocks become input to the message digest processing logic.
- This algorithm has 4 rounds, each round consists of 16 steps. Each round takes the current 512 bit block, the register abcdef and a constant K[t](where t=0 to 63) as the three inputs.
- Here we have 4 constants defined for K[t],one used in each of the 4 rounds. The values of K[t] are shown in the Table 1.

Table 1. Values of K[t]

Round	Value of 't' between	K[t] in hexa-decimal
1	1 and 16	0x5A927999
2	17 and 32	0x6ED9EBA1
3	33 and 48	0x9F1BBCDC
4	49 and 64	0XCA62C1D6

Each round containing 16 iterations this makes it a total of 64 iterations. Aniteration consists of following operations:

Wt = Mti 0<t<16.

The values of Wt are calculated as follows:

For the first 16 words of W(i.e t=0 to 15), the contents of the input message sub block M[t] become the contents of W[t]. The first 16 blocks of the input message M are copied to W. The remaining values of W are derived using equation:

w[i] = (w[i-3] ***xor*** *w[i-8]* ***xor*** *w[i-14]* ***xor*** *w[i-16])* (3.1)

F is the logical operations performed in each round. Process is divided in four rounds and each round contains sixteen transformation steps. Values of constant using in this 64 rounds is from i=0 to 63.

The pseudo code for each round which consists of 16 transformation steps is given below:

if 0 ≤ i ≤ 15 then

f = (b and c) or ((not b) and d) K[t] = 0x5A827999

P =(b and c) or ((not b) and d) if 16 ≤ i ≤ 31

f = b xor c xor d K[t] = 0x6ED9EBA1

p = (d and b) or ((not d) and c); if 32 ≤ i ≤ 47

f = (b and c) or (b and d) or (c and d) K[t] = 0x8F1BBCDC

p = (b and c) and d if 48 ≤ i ≤ 63

f = b xor c xor d K[t] = 0xCA62C1D6

p= c and (b or (not d))

Initialize the six chaining variables with (t-1) hash value. for t=0 to 63

{

T =ROTL2 (A) + f (B, C, D) +E + Kt + Wt; L =ROTL2 (A) + f (B, C, D) +E +F +Kt+ Wt; F=T; E= D; D= C;
C= ROTL15(B); B= ROTL5 (A); A=L;

F=E; E=D; D=C; C=B;

B= A+p(B,C,D)+Mi+tk+S+B; A=F;

}

Where tk= t[k] is the constant used in second block, s is the number of left rotation to be executed, f is the coefficients of each round is a bitwise Boolean function that is used in first block, p is a bitwise Boolean function that is used in second block.

The values of constant t[k] are given as:

t[k] from [0..3] := { 0xd76aa478, 0xe8c7b756, 0x242070db, 0xc1bdceee }, t[k] from [4.. 7] := { 0xf57c0faf, 0x4787c62a, 0xa8304613, 0xfd469501 }, t[k] from [8..11] := { 0x698098d8, 0x8b44f7af, 0xffff5bb1, 0x895cd7be }, t[k] from [12..15] := { 0x6b901122, 0xfd987193, 0xa679438e, 0x49b40821 }, t[k] from [16..19] := { 0xf61e2562, 0xc040b340, 0x265e5a51, 0xe9b6c7aa }, t[k] from [20..23] := { 0xd62f105d, 0x02441453, 0xd8a1e681, 0xe7d3fbc8 }, t[k] from [24..27] := { 0x21e1cde6, 0xc33707d6, 0xf4d50d87, 0x455a14ed }, t[k] from [28..31] := { 0xa9e3e905, 0xfcefa3f8, 0x676f02d9, 0x8d2a4c8a }, t[k] from [32..35] := { 0xfffa3942, 0x8771f681, 0x6d9d6122, 0xfde5380c }, t[k] from [36..39] := { 0xa4beea44, 0x4bdecfa9, 0xf6bb4b60, 0xbebfbc70 }, t[k] from [40..43] := { 0x289b7ec6, 0xeaa127fa, 0xd4ef3085, 0x04881d05 }, t[k] from [44..47] := { 0xd9d4d039, 0xe6db99e5, 0x1fa27cf8, 0xc4ac5665 }, t[k] from [48..51] := { 0xf4292244, 0x432aff97, 0xab9423a7, 0xfc93a039 }, t[k] from [52..55] := { 0x655b59c3, 0x8f0ccc92, 0xffeff47d, 0x85845dd1 }, t[k] from [56..59] := { 0x6fa87e4f, 0xfe2ce6e0, 0xa3014314, 0x4e0811a1 } and t[k] from [60..63] := { 0xf7537e82, 0xbd3af235, 0x2ad7d2bb, 0xeb86d391 }.

S specifies per round shift amount and its values are given below. s[0..15] := {7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22}, s[16..31] := {5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20}, s[32..47] := {4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23} and s[48..63] := {6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21}.

Round Addition

Round addition computes the summation of round logic output with the initial Hash values to obtain the final Hash output which is of 192-bits. The intermediate Hash values are computed for larger length messages which are given below.

$H0^{(i)} = A + H0^{(i-1)}$,

$H1^{(i)} = B + H1^{(i-1)}$,

$H2^{(i)} = C + H2^{(i-1)}$,
$H3^{(i)} = D + H3^{(i-1)}$,

$H4^{(i)} = E + H4^{(i-1)}$ and

$H5^{(i)} = F + H5^{(i-1)}$.

The output transformation step is modular summation used to map the final output of the single compression function of n bits to the output length.

SIMULATION RESULTS

Simulation results of Round Logic

The input to the Round Logic is ‘clk’, ‘rst’, and the input ‘w[t]’ is of 32 bit size, output is ‘shaout1’ with 192 bit size. The output is obtained according to round logic operation.

Figure 1. Simulation results of Round Logic

Simulation Results of Enhanced SHA-192

The input to the Enhanced SHA-192 Algorithm is ‘clk’, ‘rst’, and the input ‘msgin’ is of 200 bit size, output is ‘shaout’ with 192 bit size. The output is obtained according to Enhanced SHA-192 Algorithm operation explained in the chapter3. Figure 5.6 shows simulation results of Enhanced SHA-192 Algorithm.

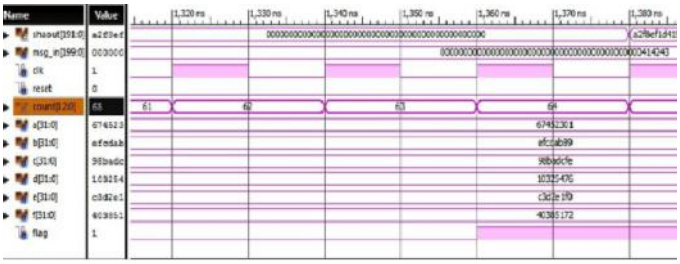


Figure 2. Simulation Results of Enhanced SHA-192

Conclusion

The Enhanced SHA-192 Algorithm is designed by using Xilinx ISE design suite 12.4 version with Verilog HDL. The design is simulated for functionality and input and output is verified by using Xilinx ISE Simulator tool.So, this algorithm can be used in high security applications like Digital signatures and Authentication. This algorithm can be used in Digital Certificate and Message integrity applications.

REFERENCES

Secure Hash Standard (SHS), 2012. Federal Information Processing Standards Publication (FIPS-PUB) 180-4, March 2012.

Garbita Gupta, Sanjay Sharma, 2013. “Enhanced SHA-192 Algorithm with Larger Bit Difference” International Conference on Communication Systems and Network Technologies.

Thulasimani, L. and Madheswaran, M. 2009. “Security and Robustness Enhancement of Existing Hash Algorithm” IEEE International Conference on Signal Processing Systems.

Eli Biham, Rafi Chen, Antoine Joux, Patrick Carribault, Christophe Lemuet, William Jalby, “Collision in SHA-0 and Reduced SHA-1,”Advances in Cryptology-EUROCRYPT 2005, LNCS 3494, SpringerVerlag,2005.

William Stallings, “Cryptography and Network Security: Principles and Practice. Fifth Edition, Pearson Hall, 2011.

HarshvardhanTiwari. A Secure Hash Function MD-192 with Modified Message Expansion” Vol.7 No.2 February 2010 *International Journal of Computer Science and Information Security*.

Florent Chabaud, Antoine Joux, “Differential collisions in SHA-0”,Advances in Cryptology-CRYPTO’98, LNCS 1462, Springer-Verlag,1998.

Matusiewicz, K. and Pieprzyk, J. 2004. “Finding good differential patterns attacks on SHA-1”, (Pub 2004), Available:http://eprint.iacr.org/2004/364.pdf

Marc Stevens hash clash - Framework for MD5 & SHA-1 Differential Path Construction and Chosen-Prefix Collisions for MD5

****