Available online at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/jisa

ASH-512: Design and implementation of cryptographic hash algorithm using co-ordinate geometry concepts

Pallipamu Venkateswara Rao [a,*], *K. Thammi Reddy* [b], *P. Suresh Varma* [a]

[a] *Department of Computer Science and Engineering, Adikavi Nannaya University, Rajahmundry 533105, Andhra Pradesh, India*

[b] *Department of Computer Science and Engineering, GITAM University, Visakhapatnam 530045, Andhra Pradesh, India*

Keywords:
Cryptography
Hash function
Message digest
Message authentication
Co-ordinate geometry concepts

ABSTRACT

Sending and receiving information over the internet is easy, fast and cost effective due to development of information technology. These days most of the information either personal or official is communicated through internet only. Most of the paper based documents are replaced by e-documents. So, there is a need to protect confidential or sensitive information in the computer as well as during transmission. Message digest is a mechanism, which is used to ensure message authentication and integrity. 'Algorithm for Secure Hashing-512' (ASH-512) is a novel algorithm proposed in this paper, which is designed using co-ordinate geometric concepts. The algorithm accepts 1024-bits as input and produces 512-bits as output. The algorithm is more secure and easy to construct and thus made this algorithm special. The algorithm is implemented in Java and the results are compared with standard cryptographic hash algorithm-SHA2 (512) and Whirlpool-512.

© 2014 Elsevier Ltd. All rights reserved.

1. Introduction

Cryptographic hash functions are the tools of cryptography to generate message digest or hash code, which is used in variety of information security applications and protocols. Message digest is the essential element in digital signature schemes and is also used in message integrity, password protection, random number generation, challenge-response protocol etc. Hash algorithm takes arbitrary length input (or empty) and produces a fixed length hash code or message digest as output. The need for novel secure hash algorithms increases proportionately with the increased demand of secure communications. The existing hash functions are actively used in various information security applications and protocols, but currently not able to prevent attacks effectively because attacks are increased due to enhanced computing power of the computers.

The MD2, MD4, RIPEMD family (Mironov, 2005), MD5 (Gauravaram et al., 2006; Wang and Yu, 2005), SHA0 (Kahate, 2006), GOST, Whirlpool, Tiger, SHA1 (Biham et al., 2005; Wang et al., 2005), SHA2 (Federal Information Processing Standards Publication, 2012; Gilbert and Handschuh, 2004) are some of the popular algorithms (Tiwari and Asawa, 2010; Brown et al., 2008) with message digest of different sizes, but their security power is reducing gradually (Ristenpart et al.,

* *Corresponding author*. Tel.: +91 9441447037.
E-mail address: venkat.aknu@gmail.com (P. Venkateswara Rao).
http://dx.doi.org/10.1016/j.jisa.2014.10.006
2214-2126/© 2014 Elsevier Ltd. All rights reserved.

Please cite this article in press as: Venkateswara Rao P, et al., ASH-512: Design and implementation of cryptographic hash algorithm using co-ordinate geometry concepts, Journal of Information Security and Applications (2014), http://dx.doi.org/10.1016/j.jisa.2014.10.006

2011; Knudsenl et al., 2007). The time complexity for finding a collision for MD4 is about 2^{23} MD4 operations without the multi-message modification, and is about 2^{8} MD4 operations with the multi-message modification (Wang and Yu, 2005). The time complexity for finding a collision for HAVAL-128 is about 2^{13} MD4 operations without the multi-message modification, and is 2^{7} HAVAL-128 operations with the multi-message modification (Wang and Yu, 2005). The time complexity for finding a collision for RIPEMD is about 2^{30} RIPEMD operations without the multi-message modification, and is 2^{18} RIPEMD operations with the multi-message modification (Wang and Yu, 2005). The time complexity for finding a collision for SHA-0 is about 2^{61} SHA-0 operations without the multi-message modification, and is 2^{45} SHA-0 operations with the multi-message modification (Wang and Yu, 2005). Stevens et al. have shown how, at an estimated cost of 2^{39} calls to the MD5 algorithm compression function, Chosen prefixes (any two) are P_x and P_y and suffixes are S_x and S_y to be constructed such that the results of concatenation of $P_x||S_x$ and $P_y||S_y$ collide under MD5 (Stevens et al., 2012). Xiaoyun et al. have published a paper on HAVAl-180, which has shown that any message of 1024 bits (m), they have modified on m and the modified message can collide with another message m_1 with the probability of $1/2^7$, wherein $m_1 = m + \Delta m$, (Δm is a fixed difference selected in advance) (Xiaoyun et al., 2005). An attack was announced by Xiaoyun et al. on SHA-1. The attack can find collisions in the SHA-1, requiring less than 2^{69} operations and a brute-force search would require 2^{80} operations (Schneier on security: SHA-1 broken). In 2005, Vincent Rijmen and Elisabeth Oswald published a paper entitled 'Update on SHA-1'. In this paper, they show that an attack on a reduced version of SHA-1, with 53 rounds out of 80 rounds, there is a possibility to find collisions less than 2^{80} operations (Rijmen and Oswald, 2005).

The National Institute of Standards and Technology (NIST) announced as a public request for new secure hash algorithm (Third-round report of the SHA-3 cryptographic hash algorithm competition, 2012; Status report on the second round of the SHA-3 cryptographic hash algorithm competition, 2011; Brown et al., 2008), which is named as SHA-3. In this contest Keccak algorithm (Status report on the first round of the SHA-3 cryptographic hash algorithm competition, 2009) has been selected as SHA-3 family (Martin, 2008) on 2012 and it is yet to be standardized. Taking NIST initiative as inspiration, we tried to design more secure hash algorithm. Section 2 of this paper denotes an overview of popular cryptographic hash functions.

Section 3 proposes a novel secure hash function (ASH-512), which is developed using co-ordinate geometry concepts. Section 4 deals with results and discussion. The security and efficiency analysis of proposed algorithm is discussed in Section 5. Finally Section 6 concludes with security strength of ASH-512.

2. Review of cryptographic hash functions design

Hash functions are currently a hot topic of research in cryptography. The area of information security welcomes new approaches to the design of secure hash functions. Innumerable hash functions (Sheena Mathew and Jacob, 2010; Knudsenl et al., 2007) have been developed and used in various security applications. The designs of some of the existing algorithms are briefly discussed below (Ferguson, 2010).

a. *Iterated hash functions*

The cryptographic hash function takes an input of arbitrary length of message and produces a fixed length of message digest or hash code as output. It is difficult to design an algorithm that accepts messages of variable length and/or gives message digest of variable length. The hash functions are designed so far based on compression function, which accepts fixed length input and produces a fixed length output. The design principle of iterated hash functions involves dividing the input into fixed length blocks and each block is passed into the compression function. The resultant algorithm is named as "Iterated hash function" (Stallings, 2003).

b. *Hash functions based on block ciphers*

In the iterated hash function, designer is concentrated on compression function. The design of compression function is not an easy task. Block cipher is an another approach to construct a compression function on available cryptographic primitive. The advantage of this design is the reusability of existing implementations in software and hardware. The hashing and encryption are required for every application and complexity of the implementation is minimized by using a block cipher. The disadvantage is that the block cipher based hash functions are less efficient than dedicated hash functions.

c. *Hash functions using modular arithmetic*

In this process modular arithmetic is used as a basic building block for design of a compression function in cryptographic hash function. The reusability of existing implementation is allowed similar to asymmetric key cryptosystem. The advantage of this mechanism is that to change the security level by changing the value of modulus (M). However, this hash function exhibits very less performance compared to block cipher based hash function.

d. *Dedicated Hash functions*

The dedicated hash functions are special functions which are designed for explicit purpose of hashing. These are designed by keeping two things in mind, which are reusability and good performance.

The hash functions of this type which have received much attention in practice are based on the MD4 algorithm. MD4 (Rivest, 1992a) was originally designed towards software implementation on 32-bit platforms. Another hash function MD5 (Wang and Yu, 2005; Rivest, 1992b) is an improved variant of MD4. Based on the principle of MDx

Please cite this article in press as: Venkateswara Rao P, et al., ASH-512: Design and implementation of cryptographic hash algorithm using co-ordinate geometry concepts, Journal of Information Security and Applications (2014), http://dx.doi.org/10.1016/j.jisa.2014.10.006

(Stallings, 2003; Rivest, 1992c) family, series of hash functions were designed, which are HAVAL (X. Wang et al., 2004), RIPEMD (Xiaoyun Wang et al., 2004; Bosselaers et al., 1997; Dobbertin et al., 1996) and SHAx (Federal Information Processing Standards Publication, 2008; NIST, 1995, 2002) families of hash functions. Other hash functions are designed for 64-bit architecture (ex., Tiger), which uses look-up tables with 8-bit input and 64-bit output bits. The WHIRLPOOL is a secure hash algorithm, which is designed based on the principle of block cipher ($b = 512$ and $k = 512$) used in the Miyaguchi-Preneel mode. PANAMA is another special hash function, which is used for hashing and stream cipher encryption. Several other dedicated hash functions (Gauravaram et al.; Preenel, 1994) are not much popular because they are more vulnerable.

3. Algorithm for secure hashing (ASH-512)

One-way property of proposed algorithm ASH-512 spontaneously matches with the literal meaning of acronym 'ASH' that is powdery residue of matter remained after burning which is irreversible. The algorithm takes a message as input with a maximum length of 2^{128} bits and produces a 512-bit message digest as output. The given input message is divided into 1024-bit blocks and each block is processed with initial vector or intermediate vector (intermediate hash code). The processing steps are as follows:

a. *Append padding bits*

The message is padded so that its length is congruent to 896 modulo 1024 (length = 896 mod 1024). Padding is always performed, even though the length of given input message is 1024 bits. Thus, the number of padding bits is in the range of 1–1024. The padding starts with '1' followed by the required number of 'zeroes' i.e. 10……0.

b. *Append length*

After padding operation, a block of 128-bits which contains length of the message (before padding or original length) is appended to the padded message. This block is treated as an unsigned 128-bit integer (most significant byte first).

c. *Initialize MD buffer*

A 512-bit buffer is used to hold intermediate and final results of the hash function. The buffer is represented as eight 64-bit registers (R0 to R7). These registers are initialized to the following hexadecimal values:

R0 = 6a 09 e6 67 f3 bc c9 08
R1 = bb 67 ae 85 84 ca a7 3b
R2 = 3c 6e f3 72 fe 94 f8 26
R3 = a5 4f f5 3a 5f 1d 36 f1
R4 = 51 0e 52 7f ad e6 82 d1
R5 = 9b 05 68 8c 2b 3e 6c 1f
R6 = 1f 83 d9 ab fb 41 bd 6b
R7 = 5b e0 cd 19 13 7e 21 79

These values are similar to the initial vector values of SHA2-512 which are standardized by Federal Information Processing Standards Publications (FIPS-PUBS). These values are stored in big-endian format, i.e., the most significant byte (MSB) of a word is copied in the low-address position.

d. *Processing message in 1024-bit blocks*

The heart of the algorithm is a compression function. This module is labeled H_{ASH} in Fig. 1 and its logic is illustrated in Fig. 2.

Fig. 1 depicts the overall processing of a message to produce a message digest or hash code. The outcome message of the first two steps (after append padding bits and append length) is an integer multiple of 1024-bits in length. The outcome message is represented as the sequence of 1024-bit blocks Y_0, Y_1, Y_2 … Y_q … Y_{L-1}, so that the total length of the expanded message is $L \times 1024$ bits (L = the number of 1024-bit blocks). Here 'K' represents the actual length of the message in bits; 'IV' is the initial vector which is used to initialize the eight 64-bit registers (R0, R1, R2, R3, R4, R5, R6 and R7). CV_1, CV_2 … CV_q … and CV_{L-1} represents carry vector which hold intermediate and final result of the hash function respectively. The algorithm takes two inputs in each round, which are 1024-bit block (Y_q) of the message and 512-bit carry vector (CV_q). At the end of the Lth stage, a 512-bit message digest is produced.

Initially, the input message is divided into 1024-bit blocks and each block is passed into the hash function along with the 512-bit initial vector. The hash function is also called Hash algorithm (H_{ASH}).

The Hash function (H_{ASH}) logic is as follows:

1. *Modification function-1 (M_{fn}-1)* converts the given 1024-bit block into modified 1024-bit block.
2. *Expansion Function-1 (E_{fn}-1)* converts the 512-bit initial vector into expanded vector (1024-bits).
3. *16-bit XOR Operation (XOR_{op})* performs XOR operation on modified 1024-bit block and expanded vector (1024-bits).
4. *Expansion Function-2 (E_{fn}-2)* expands 1024-bits to 1536-bits.
5. *Splitting operation (S_{op})* each 96-bit block obtained from the above step is divided into two equal sub-blocks.
6. *Area calculation function (A_{fn})* computes area of a triangle.

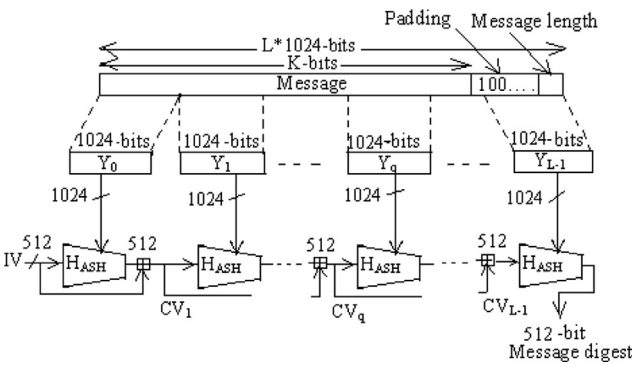

Fig. 1 – Message digest generation using ASH-512.

Please cite this article in press as: Venkateswara Rao P, et al., ASH-512: Design and implementation of cryptographic hash algorithm using co-ordinate geometry concepts, Journal of Information Security and Applications (2014), http://dx.doi.org/10.1016/j.jisa.2014.10.006

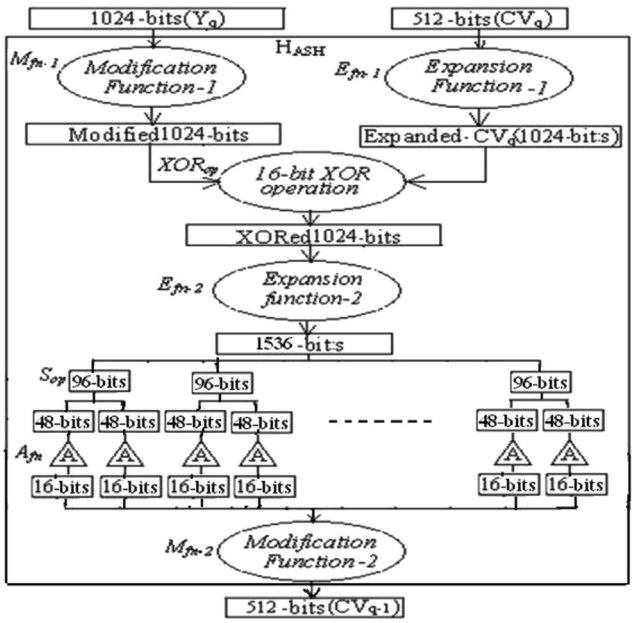

Fig. 2 – The logic of hash function.

7. *Modification function-2 (M_{fn}-2)* modifies the result of step 5.
1. *Modification function-1 (M_{fn}-1)*

Each 1024-bit block of input is divided into 128 sub-blocks comprising 8-bits in each sub block. One temporary array of size 8 (Temp8 []) is taken and initialized with zeroes. The modification function consists of two sub functions as shown in Figs. 3 and 4.

Subfunction-1: XOR operation is applied to first 8-bit sub-block of input message with Temp8 [] and then the resultant value is copied in to Temp8 []. Next the second 8-bit sub-block is XORed with Temp8 [] and again the resultant value is copied into Temp8 []. This process is continued until the message of the 1024-bit block exhausts.

Subfunction-2: The above result of Temp8 [] is incremented by 1 every time when performing XOR operation on each 8-bit unit of 1024-bit block with Temp8[], which generates a modified 1024-bit block.

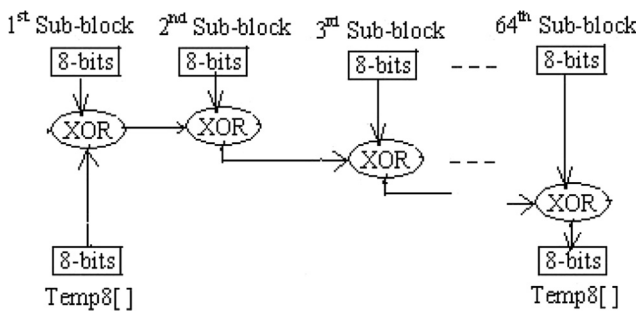

Fig. 3 – Operation of Subfunction-1.

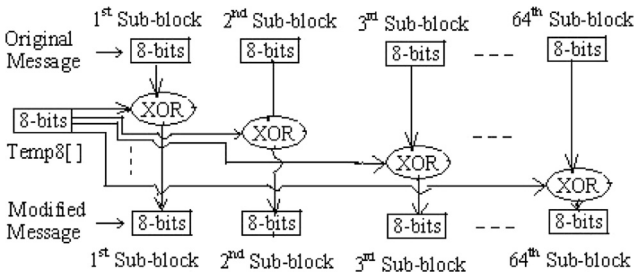

Fig. 4 – Operation of Subfunction-2.

2. *Expansion function-1 (E_{fn}-1)*

The 512-bit Initial Vector (IV) is one of the two inputs of Hash function (H_{ASH}). The contents of the registers are copied from R0 to R7 in two cycles into an array (EV1024 []), so that the total number of bits becomes equal to 1024. The result is called Expanded Vector. The concatenation process of contents of registers is shown in Fig. 5.

R0||R1||R2||R3||R4||R5||R6||R7||R0||R1||R2||R3||R4||R5||R6||R7
(|| = Concatenation operator).

3. *16-bit XOR operation (XOR_{op})*

XOR operation is performed on first 8-bits of modified 1024-bit block and first 8-bits of expanded vector. Then the result is copied into first 8-bit positions of an array (XORM1024[]) of size 1024 (Fig. 6). Again XOR operation is performed on second 8-bits of modified 1024-bit block and second 8-bits of expanded vector, and the result obtained is copied as before into next 8-bit positions of the array. This process is continued until the exhaustion of modified 1024-bit block and expanded vector.

4. *Expansion function-2 (E_{fn}-2)*

Result of the above step (*XOR operation*) is divided in to 16 equal sub blocks so that each sub block consists of 64 bits.

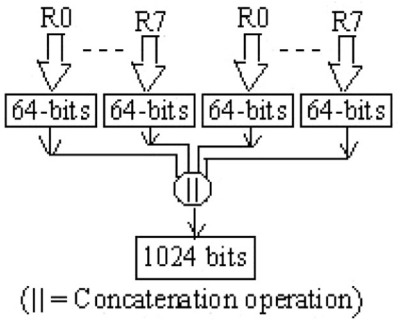

Fig. 5 – Expansion of 512-bit block (IV) to 1024-bit block (EV).

Please cite this article in press as: Venkateswara Rao P, et al., ASH-512: Design and implementation of cryptographic hash algorithm using co-ordinate geometry concepts, Journal of Information Security and Applications (2014), http://dx.doi.org/10.1016/j.jisa.2014.10.006

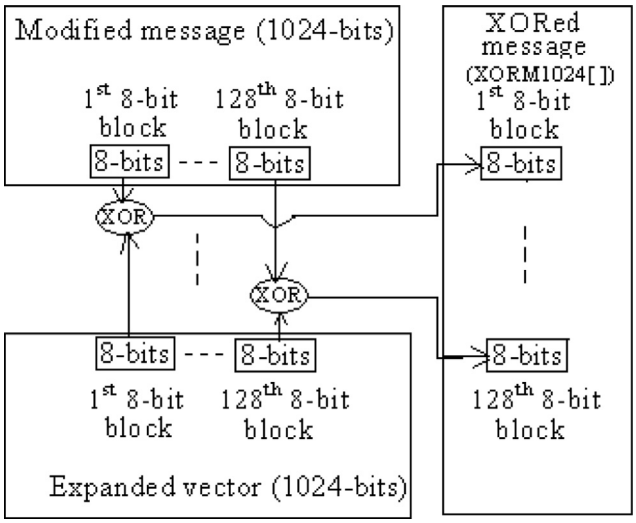

Fig. 6 – XOR operation on modified message (1024-bits) and expanded vector (1024-bits).

Expansion table (E-table) of the Data Encryption Standard (DES) is applied to each sub block twice to produce 80 bits first time and 96 bits second time (Tables 1(A)–(C)). The same process is applied to remaining sub-blocks.

Table 1(A) – Positions of 64 bits.

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

Table 1(B) – Expansion permutation-1.

63	0	1	2	3	4	5	6	7	8
7	8	9	10	11	12	13	14	15	16
15	16	17	18	19	20	21	22	23	24
23	24	25	26	27	28	29	30	31	32
31	32	33	34	35	36	37	38	39	40
39	40	41	42	43	44	45	46	47	48
47	48	49	50	51	52	53	54	55	56
55	56	57	58	59	60	61	62	63	0

Table 1(C) – Expansion permutation-2.

62	63	0	1	2	3	4	5	6	7	8	9
6	7	8	9	10	11	12	13	14	15	16	17
14	15	16	17	18	19	20	21	22	23	24	25
22	23	24	25	26	27	28	29	30	31	32	33
30	31	32	33	34	35	36	37	38	39	40	41
38	39	40	41	42	43	44	45	46	47	48	49
46	47	48	49	50	51	52	53	54	55	56	57
54	55	56	57	58	59	60	61	62	63	0	1

Bit positions are arranged as per plan shown in Tables 1(A)–(C). The above process is applied for all remaining sub blocks.

5. *Splitting operation (S_{op})*

Each 96-bit block obtained from the above step is divided into 2 blocks of 48-bits each in length, which is shown in Fig. 10.

6. *Area calculation function (A_{fn})*

Again each 48-bit block is further divided into 3 blocks each with 16 bits. The main objective is that every 16-bit block acts as a point in the co-ordinate geometric graph, i.e. first 8-bits act as *X*-axis value and second 8-bits act as *Y*-axis value. So, every 48-bit block is divided into three points of a triangle as *x*1, *y*1; *x*2, *y*2; and *x*3, *y*3 as shown in Fig. 7. These points are converted into integers before calculating area of a triangle.

7. *Modification Function-2 (M_{fn}-2)*

The resultant area of triangle is converted into binary numbers. If the bits are less than 16, zeroes are added on left hand side of the result. This 16-bit block is divided into 4 sub-blocks of 4-bits each in length. XOR operation is performed on each 4-bit sub-block with temporary array (Temp4[]) of size 4 that was already initialized with zeroes. The same process is followed for 2nd, 3rd and 4th sub-blocks. Again XOR operations are performed on Temp4[] with the previous 4-bit sub-blocks. The resultant 16 bits are converted into haxadecimal numbers which becomes part of the message digest. The same procedure is continued for all resultant areas.

e. *Output*

The 512-bit message digest is obtained by concatenating the results of all triangle areas in the form of hexadecimal numbers.

Pseudo code of the proposed algorithm, ASH-512 is described in Table 2.

All hexadecimal numbers are copied into carry vector in order, which becomes the message digest of the given message, if it is single 1024-bit block. Otherwise addition modulo-512 operation is performed with previous carry vector and the result is used as input to the hash function along with next

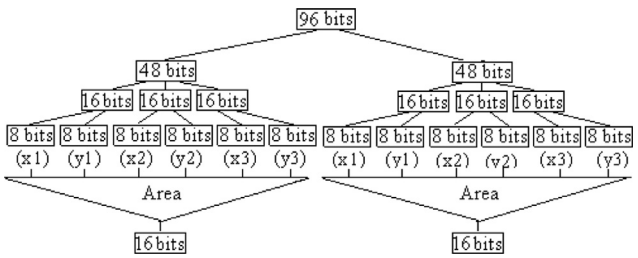

Fig. 7 – Reduction of 96 bits into two units of 16 bits each.

Please cite this article in press as: Venkateswara Rao P, et al., ASH-512: Design and implementation of cryptographic hash algorithm using co-ordinate geometry concepts, Journal of Information Security and Applications (2014), http://dx.doi.org/10.1016/j.jisa.2014.10.006

Table 2 – ASH-512 pseudo code.

Pseudo code	*Description*
Process the message in successive 1024-bit blocks: break message into 1024-bit blocks Begin for each block; break block into one twenty eight 8-bit big-endian blocks b[i], $0 \le i \le 128$ call the following functions in-order. *i. Modification function (M_{fn-1})* *ii. Expansion Function (E_{fn-1})* *iii.16-bit XOR operation (XOR_{op})* *iv. Expansion Function (E_{fn-2})* *v. Splitting operation (S_{op})* *vi. Area calculation function (A_{fn})* *vii. Modification function (M_{fn-2})*	Begin main program b[0]represent 1st 8-bit block, b[1] represent 2nd 8-bit block and so on(upto 64, 8-bit blocks)
Modification function (M_{fn-1})	*Description*

XOR operation is performed on every 8-bit block of the message:
Temp8[]={0,0,0,0,0,0,0,0};k=0;
for j from 0 to 127
for i from 0 to 7
Temp8[i]=Temp8[i] XOR b[k+i];
end loop
k=k+8
end loop
 	
k=0;
for j from 0 to 127
for i from 0 to 7
modifiedmsg[k+i]=Temp8[i] XOR b[k+i];
each time Temp8[i] value is incremented by 1
end loop
k=k+8
end loop
	Temp8[] is a temporary array of size 8 and initialized with 0's;
	
	The variables i, j and k are declared as integers;
	
	modifiedmsg is an array of size 1024;

Expansion Function (E_{fn-1})	*Description*
Extend the 512-bit MD buffer into 1024-bit block: Contents of the registers (R0 to R7) are copied circularly into an array of size 1024 (EV1024	EV1024 is an array of size 1024
[]) until total number of bits are equal to 1024.	
16 bit XOR operation (XOR_{op})	*Description*
XOR operation is performed on each 16-bit block of modifiedmsg[] and EV1024[].	Both array sizes are 1024
Expansion Function (E_{fn-2})	*Description*
The result of above step is divide into 16 equal sub-blocks. Each sub-block consists of 64-bits in length. Apply twice the operation of Extension box of DES on 1st sub-block. So, the number of bits are increases from 64 to 80 and then 80 to 96 bits. This same process is applied to the remaining sub-blocks.	Expand the above result from 1024-bit block into 1536-bit block:
Splitting operation (S_{op})	*Description*
Each 96-bit block, obtained from the above step is divided into two equal sub-blocks.	Each sub-block consists of 48 bits;
Area calculation function (A_{fn})	*Description*
Every 48-bit block is divided into three sub-blocks of 16-bits each. These three sub-blocks are act as three points of a triangle which are used in area calculation as discussed in the example (Area calculation function). The following equations are used in area calculation: $a=sqrt((x2-x1)^2+(y2-y1)^2)$ $b=sqrt((x2-x1)^2+(y2-y1)^2)$ $c=sqrt((x2-x1)^2+(y2-y1)^2)$ $s=(a+b+c)/2$ $area=sqrt((s*(s-a)+(s-b)+(s-c))$ Where, the parameters a, b, and c are sides and s is semi-perimeter of a triangle.	A1[] is an array of size 16; Temp4 is an array of size four; $A^1$1[] is an array of size 16 which contains modified values of A1[]; a1[] is an array of size 32, which contain the results of areas;
Modification function(M_{fn-2})	*Description*

The result of triangle area is converted into binary form and copied in an array (A1[]) of size 16, MSB positions are filled by zeroes if the length is less than 16 bits.
XOR operation is performed on every 4 bits of the above 16-bit block:
Temp4[]={0,0,0,0 };
k=0;
for j from 0 to 15
for i from 0 to 3
Temp4[i]=Temp4[i] XOR a1[k+i];
end loop
k=k+4;
end loop
k=0;
for j from 0 to 15
for i from 0 to 3
 	
modifieda1[k+i]=Temp4[i] XOR a1[k+i]
end loop
k=k+4
end loop
 	
The modified array ($A^1$1[]) values are converted into hexadecimal numbers.
	Temp4[] is a temporary array of size 4 and initialized with 0's;
	
	The variables i, j and k are declared as integers;

Repeat the above two functions (A_{fn}) and (M_{fn}-2) for the remaining 48-bit sub-blocks. End	End of the main program

Please cite this article in press as: Venkateswara Rao P, et al., ASH-512: Design and implementation of cryptographic hash algorithm using co-ordinate geometry concepts, Journal of Information Security and Applications (2014), http://dx.doi.org/10.1016/j.jisa.2014.10.006

1024-bit block. The above process is repeated until the last 1024-bit block.

Example:

Input string = "The quick brown fox jumps over the lazy dog."
Number of characters = 36
Total bits = 288

a) Before padding:

0 0 1 0 1 0 1 0 0 0 0 1 0 1 1 0 1 0 1 0 0 1 1 0 1 0 0 0 1 1 1 0 1
0 1 0 1 1 1 0 1 0 0 1 0 1 1 0 1 1 0 0 0 1 1 0 1 1 0 1 0 1 1 0 0 1
0 0 0 1 1 0 0 1 0 0 1 1 1 0 1 1 1 1 0 1 1 0 1 1 1 0 1 1 1 0 0 1 1
1 0 1 1 0 0 1 1 0 0 1 1 0 1 1 1 1 0 1 1 0 0 0 0 1 1 1 1 0 0 1 0 1
0 1 1 0 1 0 1 0 1 1 1 0 1 0 1 1 0 1 1 0 0 0 0 0 1 1 1 0 1 1 0 0 1
1 1 0 1 1 1 1 0 1 1 0 0 1 1 0 1 1 1 0 1 0 1 0 0 1 1 0 0 1 0 0 1 1
1 0 0 0 1 0 1 1 1 0 0 0 0 1 0 1 1 0 1 0 1 0 0 1 1 0 0 0 1 1 0 1 1
0 1 0 0 0 0 1 1 0 0 1 0 1 1 1 1 0 1 0 0 1 1 1 1 0 0 0 1 0 0 1 1 0
1 1 1 1 0 1 1 0 1 1 1 0 0 1 1 0 0 1 1 1 0 1 0 0

b) After padding:

0 0 1 0 1 0 1 0 0 0 0 1 0 1 1 0 1 0 1 0 0 1 1 0 1 0 0 0 1 1 1 0 1
0 1 0 1 1 1 0 1 0 0 1 0 1 1 0 1 1 0 0 0 1 1 0 1 1 0 1 0 1 1 0 0 1
0 0 0 1 1 0 0 1 0 0 1 1 1 0 1 1 1 1 0 1 1 0 1 1 1 0 1 1 1 0 0 1 1
1 0 1 1 0 0 1 1 0 0 1 1 0 1 1 1 1 0 1 1 0 0 0 0 1 1 1 1 0 0 1 0 1
0 1 1 0 1 0 1 0 1 1 1 0 1 0 1 1 0 1 1 0 0 0 0 0 1 1 1 0 1 1 0 0 1
1 1 0 1 1 1 1 0 1 1 0 0 1 1 0 1 1 1 0 1 0 1 0 0 1 1 0 0 1 0 0 1 1
1 0 0 0 1 0 1 1 1 0 0 0 0 1 0 1 1 0 1 0 1 0 0 1 1 0 0 0 1 1 0 1 1
0 1 0 0 0 0 1 1 0 0 1 0 1 1 1 1 0 1 0 0 1 1 1 1 0 0 0 1 0 0 1 1 0
1 1 1 1 0 1 1 0 1 1 1 0 0 1 1 0 0 1 1 1 0 1 0 0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0

c) After append length:

0 0 1 0 1 0 1 0 0 0 0 1 0 1 1 0 1 0 1 0 0 1 1 0 1 0 0 0 1 1 1 0 1
0 1 0 1 1 1 0 1 0 0 1 0 1 1 0 1 1 0 0 0 1 1 0 1 1 0 1 0 1 1 0 0 1
0 0 0 1 1 0 0 1 0 0 1 1 1 0 1 1 1 1 0 1 1 0 1 1 1 0 1 1 1 0 0 1 1
1 0 1 1 0 0 1 1 0 0 1 1 0 1 1 1 1 0 1 1 0 0 0 0 1 1 1 1 0 0 1 0 1
0 1 1 0 1 0 1 0 1 1 1 0 1 0 1 1 0 1 1 0 0 0 0 0 1 1 1 0 1 1 0 0 1
1 1 0 1 1 1 1 0 1 1 0 0 1 1 0 1 1 1 0 1 0 1 0 0 1 1 0 0 1 0 0 1 1
1 0 0 0 1 0 1 1 1 0 0 0 0 1 0 1 1 0 1 0 1 0 0 1 1 0 0 0 1 1 0 1 1
0 1 0 0 0 0 1 1 0 0 1 0 1 1 1 1 0 1 0 0 1 1 1 1 0 0 0 1 0 0 1 1 0
1 1 1 1 0 1 1 0 1 1 1 0 0 1 1 0 0 1 1 1 0 1 0 0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0
0

c.1) *Modification function-1* (M_{fn}*-1*):

Temp8[] = {0,0,0,0,0,0,0,0}; this is a temporary array and initialize with zeroes

XOR operation is performed on each 8-bit block of above step with Temp8 [];
After completion of XOR operation Temp8[] contains: Temp8 [] = {1,1,1,0,0,1,0,0};
$(11100100)_2 = (228)_{10}$//temp8[] bits converted into decimal

Again XOR operation is perform on each 8-bit block of above result with Temp8[] = {1,1,1,0, 0,1,0,0} = $(228)_{10}$ each time the Temp8 [] value is incremented by 1, that is 228, 229 … etc.

{0 0 1 0 1 0 1 0} ^ {1 1 1 0 0 1 0 0}={1 1 0 0 1 1 1 0}
1st 8-bits $(228)_{10}$

{0 0 0 1 0 1 1 0}^{1 1 1 0 0 1 0 1}={1 1 1 1 0 0 1 1}
2nd 8-bits $(229)_{10}$

{1 0 1 0 0 1 1 0}^{1 1 1 0 0 1 1 0}={0 1 0 0 0 0 0 0}
3rd 8-bits $(230)_{10}$

...

Modified message: Modified1024[]={1 1 0 0 1 1 1 0 1 1 1 1 0 0 1 1 ……………0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 };

c.2) *Expansion function-1 (E_{fn}-1)*:

Concatenating initial vector register values in the following fashion (concatenate itself – R0||R1||R2||R3||R4||R5||R6||R7||R0||R1||R2||R3||R4||R5||R6||R7):

0 1 1 0 1 0 1 0 0 0 0 0 0 1 0 0 1 1 1 1 0 0 1 1 0 0 1 1 0 0 1 1 1 1 1
1 1 1 1 0 0 1 1 1 0 1 1 1 1 0 0 1 1 0 0 1 0 0 1 0 0 0 0 0 1 0 0 0 1 0
1 1 1 0 1 1 0 1 1 0 0 1 1 1 1 0 1 0 1 0 1 1 1 0 1 0 0 0 0 1 0 1 1 0 0
0 0 1 0 0 1 1 0 0 1 0 1 0 1 0 1 0 0 1 1 1 0 0 1 1 1 0 1 1 0 0 1 1 1
1 1 0 0 0 1 1 0 1 1 1 0 1 1 1 1 0 0 1 1 0 1 1 1 0 0 1 0 1 1 1 1 1
1 1 0 1 0 0 1 0 1 0 0 1 1 1 1 1 0 0 0 0 0 1 0 1 0 1 1 1 0 1 0 0 1
0 1 0 1 0 0 1 1 1 1 1 1 1 1 0 1 0 1 0 0 1 1 1 0 1 0 0 1 1 1 1 0 1 1 1
1 0 0 0 1 1 1 0 1 0 0 1 1 0 1 1 0 1 1 1 1 0 0 0 0 1 0 1 0 1 0 0 0 1
0 0 0 0 1 1 1 0 0 1 0 1 0 0 1 0 1 0 1 0 0 1 1 1 1 1 1 1 0 1 0 1 0 1 0
0 0 0 1 1 1 0 1 1 0 0 1 1 0 1 0 1 0 1 1 1 1 1 1 0 1 1 0 1 0 0 0
0 0 1 1 1 1 0 1 1 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0
0 0 1 1 1 0 1 1 0 0 1 1 0 1 0 1 0 1 1 1 1 1 1 1 0 1 1 0 1 0 0 0
0 0 1 1 0 1 0 1 1 1 0 1 0 1 1 0 1 0 1 1 0 1 0 1 1 0 1 1 1 1 1 1 0 0 0
0 0 1 1 0 0 1 1 0 1 0 0 0 1 1 0 0 1 0 0 0 1 0 0 1 1 0 1 1 1 1 1 1 1
0 0 1 0 0 0 0 1 0 1 1 1 1 0 0 1 0 1 1 0 1 0 1 0 0 0 0 0 1 0 0 1
1 1 1 0 0 1 1 0 0 1 1 0 0 1 1 1 1 1 1 1 0 0 1 1 1 0 1 0 1 1 1 1 0 0 1
1 0 0 1 0 0 1 0 0 0 0 1 0 0 0 1 0 1 1 1 0 1 1 0 1 1 0 0 1 1 1 1 0
1 0 1 1 1 0 1 0 0 0 0 1 0 1 1 0 0 0 0 1 0 0 1 1 0 0 1 0 1 0 1 0 1
0 0 1 1 1 0 0 1 1 1 0 1 1 0 0 1 1 1 1 0 0 0 1 1 0 1 0 1 1 1 0 1 1 1 1
0 0 1 1 0 1 1 1 0 0 1 0 1 1 1 1 1 1 1 0 1 0 0 1 0 1 0 0 1 1 1 1 1
0 0 0 0 0 1 0 1 0 1 1 1 0 1 0 0 1 0 1 0 1 0 0 1 1 1 1 1 1 1 1 0 1
0 1 0 0 1 1 1 0 1 0 1 0 1 0 1 1 1 1 0 0 0 1 1 1 0 1 1 0 0 1 1 1 1 0 1 0
0 1 1 1 1 0 0 0 1 0 1 0 1 0 0 0 1 0 0 0 0 1 1 1 0 0 1 0 1 0 0 1 0
0 1 1 1 1 1 1 1 1 0 1 0 1 0 1 1 0 1 1 1 1 0 0 1 1 0 1 0 0 0 0 0 0 1 0 1
1 0 1 0 0 0 1 1 0 0 1 1 0 1 1 0 0 0 0 0 1 0 1 0 1 1 0 1 0 0 0 0 1 0
0 0 1 1 0 0 0 0 0 1 0 1 0 1 1 0 0 1 1 1 1 1 1 0 0 1 1 0 1 0 1 1 0 0 0 0 0
1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 0 0 0 0 0 0 1 1 1 1 0 1 1 0 0 1 1 0 1 0
1 0 1 1 1 1 1 1 1 0 1 1 0 1 0 0 0 0 0 1 1 0 1 1 1 1 0 1 0 1 1 0 1
0 1 1 0 1 0 1 1 0 1 1 1 1 1 1 0 0 0 0 0 1 1 0 0 1 1 0 1 0 1 0 0 0 1 1 0
0 1 0 0 0 1 0 0 1 1 0 1 1 1 1 1 1 1 0 0 0 1 0 1 1 1 1 1 1 0
0 1

The resultant vector is called Expanded Vector which is explained in Section 2.

c.3) *16-bit XOR operation (XOR_{op})*:

Perform XOR operation on every 16-bits of modified message with corresponding 16-bits of expanded vector.

Modified message — Expanded vector

1st 16-bits — 1st 16-bits

{0100111001110011}^{0000100101101010}
={0100011100011001}

2nd 16-bits — 2nd 16-bits

{0101010101000000}^{0110011111100110}
={0011001010100110}

…

64nd 16-bits — 64nd 16-bits

{0000010010000000}^{0111100100100001}
={0111110110100001}

c.4) *Expansion function-2 (E_{fn}-2)*:

The result of above step is divided in to 16 equal sub blocks and thereby each sub-block contains 64 bits. Expansion table (E-table) of the Data Encryption Standard (DES) is applied to each sub block twice to produce 80 bits first time and 96 bits second time (Table 3). The same process is applied to remaining 64-bit sub-blocks.

For example the first 64-bits of the above step is expanded to 96 bits as shown below:

c.5) *Splitting operation (S_{op})*:

The resultant value (96 × 16 bits = 1536 bits) of the above step is divided into 32 sub-blocks of each 48 bits in length.

c.6) *Area calculation function (A_{fn})*:

Area of the triangle is calculated for each 48-bit block, which is shown below:

Division of 48 bits is depicted in Fig. 8.

The binary values (x1, y1), (x2, y2) and (x3, y3) are converted into integers (Fig. 9).

$$a = \mathrm{sqrt}\left((x2-x1)^2 + (y2-y1)^2\right)$$
$$= \mathrm{sqrt}\left((137-166)^2 + (153-51)^2\right) = 106.04$$

$$b = \mathrm{sqrt}\left((x3-x2)^2 + (y3-y2)^2\right)$$
$$= \mathrm{sqrt}\left((85-137)^2 + (50-153)^2\right) = 115.38$$

$$c = \mathrm{sqrt}\left((x1-x3)^2 + (y1-y3)^2\right)$$
$$= \mathrm{sqrt}\left((166-85)^2 + (51-50)^2\right) = 81.00$$

$$s = (a+b+c)/2 = 151.21$$

$$\text{Area } (A1) = \mathrm{sqrt}(s(s-a)*(s-b)*(s-c)) = 4146$$

Equivalent binary value of the resultant area is '1000000110010'.

Similarly find out the remaining areas of 48-bit blocks.

Table 3 – Expansion of 64-bits to 96-bits.

First 64-bit block	E-table is applied on 64-bit block produces 80-bits	E-table is applied on 80-bit block produces 96-bits
10011000	0100110001	101001100011
11100010	0111000100	001110001001
01100101	0011001010	100110010101
01001100	1010011001	010100110010
10100101	0101001011	001010010111
11101111	1111011111	011110111111
11010001	1110100011	111101000111
11000010	1110000101	011100001010

Please cite this article in press as: Venkateswara Rao P, et al., ASH-512: Design and implementation of cryptographic hash algorithm using co-ordinate geometry concepts, Journal of Information Security and Applications (2014), http://dx.doi.org/10.1016/j.jisa.2014.10.006

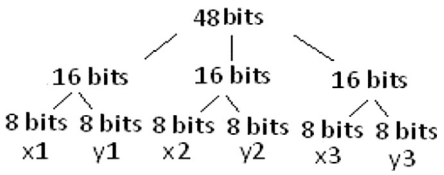

Fig. 8 – Division of 48 bits.

c.7) *Modification function-2 (M_{fn}-2)*:

Above result is modified (Area (A1)) as follows:

Temp4[] = {0,0,0,0}; this is a temporary array of size 4 and initialized with zeroes.

XOR operation is performed on each 4-bit block of Area (A1) with Temp4[];

{0000}{0001} = {0001}
{0001}{0000} = {0001}
{0001}{0011} = {0010}
{0010}{0010} = {0000}

Temp4[] = {0,0,0,0}; this is modified Temp4[];

Once again XOR operation is performed on each 4 bit block of Area (A1) with modified Temp4[];

{0001}{0000} = {0001}
{0000}{0000} = {0000}
{0011}{0000} = {0011}
{0010}{0000} = {0010}

Modified Area ($A^1$1) = {0 0 1 0 0 0 1 1 0 0 0 0 0 0 0 1}

Similarly the remaining modified Areas ($A^1$1, $A^2$2, …) were also found out.

Modified Area ($A^1$1) = {0010001100000001} = {2301}
Modified Area ($A^2$2) = {1110101101100111} = {eb67}
Modified Area ($A^3$3) = {0110001101110011} = {6373}
Modified Area ($A^4$4) = {1111110100110010} = {fd32}
Modified Area ($A^5$5) = {1110011010011001} = {e699}
Modified Area ($A^6$6) = {0110111010001000} = {6e88}
Modified Area ($A^7$7) = {0101101111111100} = {5bfc}
Modified Area ($A^8$8) = {1000011011101111} = {86ef}
Modified Area ($A^9$9) = {0000100110010100} = {0994}
Modified Area (A^{10}10) = {0110110010110100} = {6cb4}
Modified Area (A^{11}11) = {0100001001111011} = {427b}
Modified Area (A^{12}12) = {1110111000110010} = {ee32}
Modified Area (A^{13}13) = {0100101111101100} = {4bec}

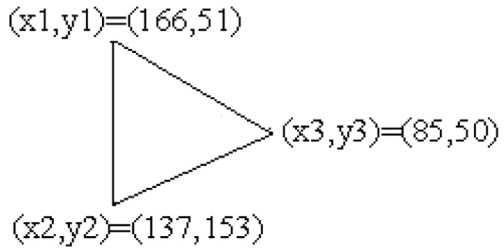

Fig. 9 – Representation of triangle points in integer form.

Modified Area (A^{14}14) = {1010001010010111} = {a297}
Modified Area (A^{15}15) = {0000110011001100} = {0ccc}
Modified Area (A^{16}16) = {0001011101001011} = {174b}
Modified Area (A^{17}17) = {1011111100000100} = {bf04}
Modified Area (A^{18}18) = {1100000111010000} = {c1d0}
Modified Area (A^{19}19) = {0001000100000001} = {1101}
Modified Area (A^{20}20) = {0100101111110001} = {4bf1}
Modified Area (A^{21}21) = { 0001110011101100} = {1cec}
Modified Area (A^{22}22) = {1101110000000110} = {dc06}
Modified Area (A^{23}23) = {0110011100010000} = {6710}
Modified Area (A^{24}24) = {1011110001011101} = {bc5d}
Modified Area (A^{25}25) = {1001100000000001} = {9801}
Modified Area (A^{26}26) = {0111010000100001} = {7421}
Modified Area (A^{27}27) = {0100001001111011} = {427b}
Modified Area (A^{28}28) = {1110111000110010} = {ee32}
Modified Area (A^{29}29) = {0100101111101100} = {4bec}
Modified Area (A^{30}30) = {1010001010010111} = {a297}
Modified Area (A^{31}31) = {0000110011001100} = {0ccc}
Modified Area (A^{32}32) = {0100000000001000} = {4008}

Message digest of a given message is obtained by concatenation of all the results of Modified Areas i.e. "2301eb 676373fd32e6996e885bfc86ef09946cb4427bee324be-ca2970ccc174bbf04c1d011014bf11cecdc066710bc5d9801742 1427bee324beca2970ccc4008".

The above hash code itself is message digest of input if it is single 1024-bit block, otherwise these values are copied into registers (carry vector) and addition modulo-512 is performed with previous carry vector and the result is used as input to the hash function along with next 1024-bit block. The above process is repeated until the last 1024-bit block to obtain message digest.

4. Results and discussions

The ASH-512, SHA2 (512) and Whirlpool-512 algorithms have been implemented in Java and run on Intel® Dual CPU 1.86 GHz, 1 GB RAM and Windows-XP. Message digest or output of the each algorithm is represented in the form of hexadecimal numbers. So, 512-bit message digest is represented in 128 hexadecimal numbers.

Outputs of ASH-512 for three sample inputs:

Input String: "The quick brown fox jumps over the lazy dog."
Output (HashCode): "2301eb676373fd32e6996e885bfc86e f09946cb4427bee324beca2970ccc174bbf04c1d011014bf11 cecdc066710bc5d98017421427bee324beca2970ccc4008"

Even a small change in the message with overwhelming probability will result in a completely different hash due to the *avalanche effect*. For example, changing *d*og to *c*og produces a hash with 125 different hexadecimal numbers out of 128 hexadecimal numbers. Changing d*o*g to e*o*g produces a hash with 123 different hexadecimal numbers out of 128 hexadecimal numbers (Fig. 10).

Input String: "The quick brown fox jumps over the lazy cog."

Please cite this article in press as: Venkateswara Rao P, et al., ASH-512: Design and implementation of cryptographic hash algorithm using co-ordinate geometry concepts, Journal of Information Security and Applications (2014), http://dx.doi.org/10.1016/j.jisa.2014.10.006

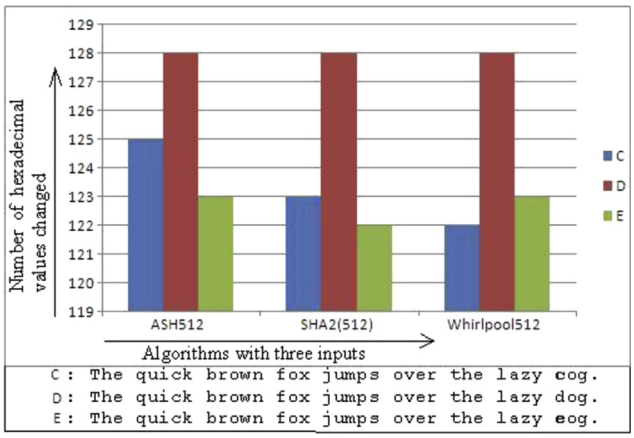

Fig. 10 – Comparison of three algorithms with three inputs (dog is replaced by cog and eog).

Output (HashCode): "5afddf28652296fa388bac6978e7918 9eb504da7725ad2fff2d60ddb28ad716f7603ee1886c3cd5ced 201ddc59c58803725b909c6ac5415c6702ad79210028b7 "
Input String: "The quick brown fox jumps over the lazy eog."
Output (HashCode): "6507d94317700eea3cd233169d650fff 624c7a9e0899c843330fc856cf00c1deca6eeb59d1e609d55370 40497f803b8c716c1fec099c9b2ae79e101f94d9862f "

Outputs of SHA2 (512) for three sample inputs:

Input String: "The quick brown fox jumps over the lazy dog."
Output (HashCode): "91ea1245f20d46ae9a037a989f54f1f790 f0a47607ee b8a14d12890cea77a1bbc6c7ed9cf205e67b7f2b8 fd4c7dfd3a7a8617e45f3c463d481c7e586c39ac1ed"
Input String: "The quick brown fox jumps over the lazy cog."
Output (HashCode): "e96eb3e153a990b69442f4ece31dee- f67494e6601516ec24cdca2542- f3117afd74bab164fc88016089a494293d42c6cca71072d9cc0- b0a522b23e253d1b51c46 "
Input String: "The quick brown fox jumps over the lazy eog."
Output (HashCode): "d7739bdc867ae3f630451108d08c- bad90d79b51d935c244162588c95- b634818592714362e7101529576cda76fc0b586d- fecd337d1d3b88a9802e6e59bd9e9818 "

In the above example, changing *d*og to *c*og produces a hash with 123 different hexadecimal numbers out of 128 hexadecimal numbers. Changing *d*og to *e*og produces a hash with 122 different hexadecimal numbers out of 128 hexadecimal numbers (Fig. 10).

Outputs of Whirlpool-512 for three sample inputs:

Input String: "The quick brown fox jumps over the lazy dog."
Output (HashCode): "87a7ff096082e3ffeb86db10feb91c5af 36c2c71bc426fe310ce662e0338223e217def0eab0b02b80eecf 875657802bc5965e48f5c0a05467756f0d3f396faba"
Input String: "The quick brown fox jumps over the lazy cog."
Output (HashCode): "f2d8d52862483e3611f1f6598291e951f fca73cc0a66dab0a3a271353e03161c93bff29539aa5b785023 cdff621ec31b961c74461f53d6eedef1cd61652bc673"
Input String: "The quick brown fox jumps over the lazy eog."
Output (HashCode): "99f6d08998623f8cbd94838b1d7aff 399c76feacd0be68838b809073f109eaf81d9442d8bada5c368 202aa27e984a4d70b67a97a4f5e4dfe9b14935db7aeb0a4".

In the above example, changing *d*og to *c*og produces a hash with 122 different hexadecimal numbers out of 128 hexadecimal numbers. Changing *d*og to *e*og produces a hash with 123 different hexadecimal numbers out of 128 hexadecimal numbers (Fig. 10).

Sample messages are given in Table 4. The messages are given in ASCII format, while the corresponding hash results are in hexadecimal format.

5. Analysis of ASH-512

a). *Security analysis*

The cryptographic hash function ASH-512 is designed with the motive of strengthening security aspects. This algorithm has achieved mainly two important goals, one is strong avalanche effect and the other is one-way property (i.e., irreversible). The One way property says that the hash code is produced from the input message but not in vice versa. The *Modification function-1* describes how a big change occurs in hash code when input message alters a little (even one bit). This is achieved by performing XOR operations as mentioned in above sections. Even though XOR is linear and doesn't prevent differential attacks, but it is used to get strong avalanche effect. Stronger security of the hash code is achieved by *Area Calculation function, Expansion functions (1 and 2) and 16-bit XOR operation*. The Area calculation function is designed based on co-ordinate geometry concepts as mentioned in Section 5. The achievement of one way property is described below:

In the output of Section 4, each 96-bit block is divided into two equal blocks of 48 bits each. The first 48-bit block is considered as: '101001100011001110001001100110010101010100110010'.

The first 48-bit block is further divided into three sub-blocks of 16-bits length each. These three sub-blocks are represented as three points of a triangle (Fig. 11) and later converted into integers before calculating the area.

Ist Sub-block is represented in three points:

1st point ($x1$, $y1$) = ((10100110), (00110011))
2nd point ($x2$, $y2$) = ((10001001), (10011001))
3rd point ($x3$, $y3$) = ((01010101), (00110010))
Area of above triangle = 0010 0011 0000 0001 = 2301

In this way we can find out areas of remaining 48-bit blocks and concatenate the results, then convert into hexadecimal form. So, it shows the one way property of ASH-512 (Fig. 12).

Please cite this article in press as: Venkateswara Rao P, et al., ASH-512: Design and implementation of cryptographic hash algorithm using co-ordinate geometry concepts, Journal of Information Security and Applications (2014), http://dx.doi.org/10.1016/j.jisa.2014.10.006

Table 4 – Comparison of ASH-512, SHA2 (512) and Whirlpool-512.

Input	ASH-512	SHA2 (512)	Whirlpool-512
" " (empty string)	0766c49b5422d6a4cb640658da761ba7e1 edfd2f7438e79e1bab45070311f0ee652cff 227613257f4ee9919f3ffd5350200279e39b2e 78e6db62550f2573e2fd	f90ddd77e400dfe6a3fcf479b00b1ee29e7015c5bb8cd70f5f15b4886cc 339275ff553fc8a053f8ddc7324f45168cffaf81f8c3ac93996f6536eef3835 e40768	99f6d08998623f8cbd94838b1d7aff399c76feacd0b e68838b809073f109eaf81d9442d8bada5c368202aa2 7e984a4d70b67a97a4f5e4dfe9b14935db7aeb0a4
"a"	406078e8b6dbbe521cecdc066710bc5d980 17421427bee324beca2970ccc1746bf04c1d0 11014bf11cecdc066710bc5d98017421427be e324beca2970ccc5bcf	1f40fc92da241694750979ee6cf582f2d5d7d28e18335de05abc54d056 0e0f5302860c652bf08d560252aa5e74210546f369fbbbce8c12cfc7957 b2652fe9a75	8aca2602792aec6f11a67206531fb7d7f0dff594131 45e6973c45001d0087b42d11bc645413aeff63a4239 1a39145a591a92200d560195e53b478584fdae231a
"abc"	2dcfb7d863430893f8604272b6c3c842be5a f3de881a642eb5d6e842624d9a3b0fdc770ac f2d5fa1ad7085f5b823c94bea549b3414456a cec6a2e798a0ab69ae	Ddaf35a193617abacc417349ae20413112e6fa4e89a97ea20a9eeee64 b55d39a2192992a274fc1a836ba3c23a3feebbd454d4423643ce80e2a9 ac94fa54ca49f	4e2448a4c6f486bb16b6562c73b4020bf3043e3a73 1bce721ae1b303d97e6d4c7181eebdb6c57e277d0e 34957114cbd6c797fc9d95d8b582d225292076d4eef5
"0123456789"	91841777615aef33b3888f743fecbd5f8b2d 0013c0c12aa13307f789dc1301336bf118882 64550501cecdc066710bc5d98017421427be e324beca2970ccc3132	bb96c2fc40d2d54617d6f276febe571f623a8dadf0b734855299b0e107 fda32cf6b69f2da32b36445d73690b93cbd0f7bfc20e0f7f28553d2a442 8f23b716e90	87d33c95622c2f2ad163c3dc715218ee7d9acadb5 c767a7fa4fe9f433cae6ca283d9a99b3a5212d33ec7 f11b154e505e2bfd8d21293e02f5030fa7e99f286f19
"message digest"	7bf2f792d868515c33388b259d451aa749f9 6dad4403829f4beca2970ccc174bbf04c1d01 1014bf11cecdc066710bc5d98017421427bee 324beca2970ccc0ae4	107dbf389d9e9f71a3a95f6c055b9251bc5268c2be16d6c13492ea45 b0199f3309e16455ab1e96118e8a905d5597b72038ddb372a8982604 6de66687bb420e7c	378c84a4126e2dc6e56dcc7458377aac838d00032 230f53ce1f5700c0ffb4d3b8421557659ef55c106b4b 52ac5a4aaa692ed920052838f3362e86dbd37a8903e
"a … z"	0ea67e90f2dccd1fd0f7f294a1bbf4b03cf03 31b3a9eec225373a82c2aac514d8a2fc1c1ed 35eb54002c8b2b8f5cc1c50cc9c3edd85cd0d 92a99ef1cc48c7ac5	4dbff86cc2ca1bae1e1b468a05cb9881c97f1753bce3619034898faa1a abe429955a1bf8ec483d7421fe3c1646613a59ed5441fb0f321389f77f48 a879c7b1f1	f1d754662636ffe92c82ebb9212a484a8d38631ead 4238f5442ee13b8054e41b08bf2a9251c30b6a0b8aa e86177ab4a6f68f673e7207865d5d9819a3dba4eb3b
"A … Z a … z 0 … 9"	Ae5da93a69e12463f78baa0eaf4713374d8 c8194a0a01dcc18b09d43d6b44403ea4a256 e9b20076a1cce312d1cd61cd3b29123069f77 743b83b10555760393bb	1e07be23c26a86ea37ea810c8ec7809352515a970e9253c26f536cfc7 a9996c45c8370583e0a78fa4a90041d71a4ceab7423f19c71b9d5a3e01 249f0bebd5894	Dc37e008cf9ee69bf11f00ed9aba26901dd7c28cd ec066cc6af42e40f82f3a1e08eba26629129d8fb7cb 57211b9281a65517cc879d7b962142c65f5a7af01467
8 times "1234567890"	93b42ee44263af75727993a7fc3f9c66176121 36264975336bd1bd63bf4d7034040d347d256 43476255ed9536710bc5d98017421427bee 324beca2970ccc38aa	72ec1ef1124a45b047e8b7c75a932195135bbb1de24ec0d1914042246 e0aec3a2354e093d76f3048b456764346900cb130d2a4fd5dd16abb5 e30bcb850dee843	466ef18babb0154d25b9d38a6414f5c08784372bc cb204d6549c4afadb6014294d5bd8df2a6c44e538c d047b2681a51a2c60481e88c5a20b2c2a80cf3a9 a083b

Please cite this article in press as: Venkateswara Rao P, et al., ASH-512: Design and implementation of cryptographic hash algorithm using co-ordinate geometry concepts, Journal of Information Security and Applications (2014), http://dx.doi.org/10.1016/j.jisa.2014.10.006

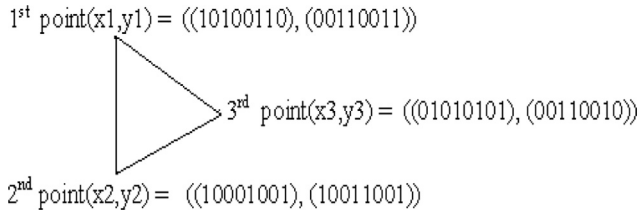

Fig. 11 – Representation of triangle with three points.

Algorithm-independent attacks:

Random attack – The probability of breaking of this algorithm is $1/2^{512}$. The number of trails and the expected values are the key parameters of this attack.
Birthday attack – Since the length of the message digest is 512 bits, the possibilities for attacks are 2^{512}. The Cryptanalyst may generate two samples from digest, which represented as S_1 and S_2. The approximate probability of two samples is as follows:

$$P \approx 1 - \left(1 \div \left(e^{2((S_1 S_2) \div 512)}\right)\right)$$

Algorithm-dependent attacks:

Meet-in-the-middle attack – The probability of middle value generation using first part samples (q_1) and last part samples (q_2) is as follows:

$$P \approx 1 - \left(1 \div \left(e^{\left((q_1 q_2) \div 2^n\right)}\right)\right)$$

Constrained meet-in-the-middle attack – This is also same as the above but uses certain constraints.
Generalized meet-in-the-middle attack – In this attack given message is divided in to $2 \cdot 10^{p-1}$ blocks (p-fold iterated scheme). To break this scheme, $10^p \cdot 2^{512/2}$ operations are required.
Correcting block attack – The attacker takes a message and its message digest and tries to change blocks several times to make sure if the digest remains same.
Differential attack – The principle behind this attack is that, if relation exists between input and output differences, collision occurs when the difference is zero.

b). *Performance Analysis*

These three algorithms ASH-512, SHA2 (512) and Whirlpool-512 were tested for comparison based on the execution time (nanoseconds). These algorithms have been implemented in Java and run on Intel®Dual CPU 1.86 GHz, 1 GB RAM and Windows-XP. The average is obtained after running each file five times (10 KB, 20 KB, 30 KB, 40 KB and 50 KB). The results of the experiment are shown in Table 5.

However the differences in execution time are minimal when the security strength of ASH-512 is considered. The security strength of the algorithm is more important than execution time because less execution time but weak security, is of less utility. The results conclude that new algorithm ASH-512 exhibits more strength in security.

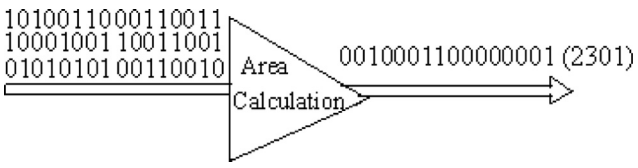

Fig. 12 – Illustration of one way property (from 48-bits to 16-bits).

c). *Construction of ASH-512 algorithm compared with the other algorithms*

The compression function of ASH-512 consists of six functions; first function is Modification function-1 (M_{fn}-1), which operates two stages. In first stage, XOR operations are performed on the temporary array (Temp8 []) and each 8-bit unit of 1024-bit block. In the second stage the resultant temporary array is XORed with each 8-bit unit of 1024-bit block. The second function (Expansion function-1 (E_{fn}-1)) includes only concatenation operations. The third function is 16-bit XOR operation (XOR_{op}), which includes exclusively XOR operations. Rearrangement of bits is done in fourth function that is Expansion function-2 (E_{fn}-2). The fifth function is Area calculation function (A_{fn}), which computes area of a triangle. The last function is Modification function-2 (M_{fn}-2), which modifies the resultant area of triangle.

The operations involved in the above functions are as follows:

i. XOR
ii. Concatenation
iii. Rearrangement of bits
iv. Calculation of triangle area

The operations involved in SHA2 (512) are as follows:

i. Addition modulo 2^{64}
ii. AND
iii. OR
iv. XOR
v. SHR (right logical shift)
vi. ROT (rotate no carry)

Whirlpool-512: The overall structure of Whirlpool-512 hash function is similar to that of block cipher, which is developed based on the features of AES algorithm.

Table 5 – Execution times of Whirlpool-512, SHA2 (512) and ASH-512 (time in nanoseconds).

File size	Algorithm		
	Whirlpool-512 Time (ns)	SHA2 (512) Time (ns)	ASH-512 Time (ns)
10 KB	76,182,308	43,420,323	71,601,558
20 KB	80,469,166	50,833,835	80,820,328
30 KB	89,164,963	39,588,830	108,773,042
40 KB	87,723,998	116,064,752	139,276,360
50 KB	78137305	125479635	167418205

Whirlpool-512 < SHA2 (512) < ASH-512.

Please cite this article in press as: Venkateswara Rao P, et al., ASH-512: Design and implementation of cryptographic hash algorithm using co-ordinate geometry concepts, Journal of Information Security and Applications (2014), http://dx.doi.org/10.1016/j.jisa.2014.10.006

Whirlpool-512 consists of the following functions:

a. Add round key
b. Substitute bytes
c. Shift columns
d. Mix rows
e. Add round key

Some of the operations involved in the above functions are given below:

i. XOR
ii. Circular right shift
iii. Multiplication
iv. Permutation
v. Mapping with S-box
vi. GF (28) polynomial ($x8 + x4 + x3 + x2 + 1$)
vii. Hamming distance = number of bit positions where bit values differ
viii. Matrix construction/orientation etc.

Number of operations involved in ASH-512 is very less when compared to SHA2 (512) and Whirlpool-512. Hence, the construction of ASH-512 is easy than other popular cryptographic hash algorithms like SHA2 (512) and Whirlpool-512.

6. Conclusion

The information security grows parallel to the growth and development of information technology and thus the message authentication and integrity services become most important in secure communication. These services are achieved by Cryptographic hash algorithms. This paper proposes a new Cryptographic hash algorithm called *Algorithm for Secure Hashing (ASH-512)*, which is designed based on Co-ordinate geometry concepts. The main strengths of ASH-512 are *Area calculation function, E-Table of DES* and *XOR operation*, which in turn result in strong nonlinear avalanche effect, increased diffusion in output and makes differential attacks difficult. The algorithm is implemented in Java and results obtained are analyzed. Based on experimental results we conclude that the proposed algorithm is more secure than SHA2 (512) and Whirlpool-512 because it exhibits strong avalanche effect. Thus it is more secure and simple than existing popular hash functions, which may make ASH-512 a substitute. Further this algorithm is useful in secure applications and protocols like digital signature, MAC, PGP, S/MIME, kerberos and SSL/TLC.

REFERENCES

Biham Eli, Chen Rafi, Joux Antoine, Carribaut Patrick, Lemuet Christophe, Jalby William. Collision in SHA-0 and reduced SHA-1. EUROCRYPT05. 2005.
Bosselaers A, Dobbertin H, Preneel B. The cryptographic hash function RIPEMD-160. CryptoBytes 1997. RSA Laboratories.
Brown D, Antipa A, Campagna M, Struik R. ECOH: the elliptic curve only hash. Submission to NIST. 2008.
Dobbertin H, Bosselaers A, Preneel B. RIPMEMD-160: a strengthened version of RIPMMD. In: Gollmann D, editor. Fast software EncrZption. LNCS, vol. 1039. Springer-Verlag; 1996.
Federal Information Processing Standards Publication. Secure hash standard (SHS). FIPS PUB 180-4. March 2012.
Federal Information Processing Standards Publication. Secure hash standard (SHS). Information Technology Laboratory, NIST; October 2008. MD 20899-8900.
Ferguson N, Lucks S, Schneier B, Whiting D, Bellare M, Kohno T, et al. The Shein Hash Function Family, Submission to NIST (Round 3). 2010.
Gauravaram Praveen, Mccullagh Adrian, Dawson Ed. Attacks on MD5 and SHA-1: is this the "sword of Damocles" for electronic commerce. March 15, 2006.
Gauravaram Praveen SS, Millan William L, May Lauren J. CRUSH: a new cryptographic hash function using iterated halving technique. GPO Box 2434, Brisbane, QLD, 4001, Australia: Information Security Research Centre, Queensland University of Technology.
Gilbert Henri, Handschuh Helena. Security analysis of SHA-256 and sisters. Lecture notes in computer science, vol. 3006; 2004. p. 175–93.
Kahate Atul. Cryptography and network security. Tata McGraw-Hill; 2006.
Knudsenl Lars R, Rechberger Christian, Thomsen Soren S. The Grindahl hash function. In: Biryukov A, editor. FSE 2007. LNCS, vol. 4593. International Association for Cryptologic Research; 2007. p. 39–57.
Martin J. Essence: a family of cryptographic hashing algorithms. Submission to NIST. 2008.
Mironov Ilya. Hash functions: theory, attacks, and applications. November 14, 2005.
NIST. Secure hash standard. FIPS PUB; 1995.
NIST. Secure hash standard. FIPS PUB 180-2. 2002.
Preenel B. Cryptographic hash functions. Trans Telecommun 1994;5:431–48.
Rijmen V, Oswald E. Update on SHA-1. LNCS, vol. 3376. Springer-Verlag; 2005. p. 58–71.
Ristenpart T, Shacham H, Shrimpton T. Careful with composition: limitations of the indifferentiability framework. In: Paterson KG, editor. Advances in cryptology – EUROCRYPT 2011-30th annual international conference on the theory and applications of cryptographic techniques, Tallinn, Estonia, May 15–19, 2011. Proceedings. Lecture notes in computer science, vol. 6632. Springer; 2011. p. 487–506.
Rivest RL. The MD5 message digest algorithm. RFC 1321. 1992.
Rivest RL. The MD4 message digest algorithm. RFC 1320. 1992.
Rivest RL. The MD2 message digest algorithm. RFC 1319. 1992.
Schneier on security: SHA-1 broken. http://www.Schneier.com/blog/archives/2005/02/sha1_broken.html.
Sheena Mathew K, Jacob Poulose. Performance evaluation of popular hash functions. World Acad Sci Eng Technol 2010;61.
Stallings William. Cryptography and network security: principles and practice. 3/e PH. 2003.
Status report on the first round of the SHA-3 cryptographic hash algorithm competition. September 2009. http://csrc.nist.gov/publications/nistir/ir7620/nistir_7620.pdf.
Status report on the second round of the SHA-3 cryptographic hash algorithm competition. February 2011. http://csrc.nist.gov/publications/nistir/ir7764/nistir-7764.pdf.
Stevens M, et al. Chosen-prefix collisions for MD5 and applications. Int J Appl Cryptogr 2012;2(4).
Third-round report of the SHA-3 cryptographic hash algorithm competition. November 2012. http://dx.doi.org/10.6028/NIST.IR.7896.
Tiwari Harshvardhan, Asawa Krishna. Cryptographic hash function: an elevated view. Eur J Sci Res 2010;43(4). ISSN: 1450-216X:452–65. © EuroJournals Publishing, Inc.

Please cite this article in press as: Venkateswara Rao P, et al., ASH-512: Design and implementation of cryptographic hash algorithm using co-ordinate geometry concepts, Journal of Information Security and Applications (2014), http://dx.doi.org/10.1016/j.jisa.2014.10.006

Wang X, Feng XD, Lai X, Yu H. Collisions for hash functions MD4, MD5, HAVAL-128 and RIPEMD. 2004. Rump session, CRYPTO 04.
Wang Xiaoyun, Feng Dengguo, Lai Xuejia, Yu Hongbo. Collisions for hash functions MD4, MD5, HAVAL-128 and RIPEMD. Revised on. August 17, 2004.
Wang Xiaoyun, Yin Yiqun Lisa, Yu Hongbo. Finding collisions in the full SHA-1. In: CRYPTO 2005 proceedings. Lecture notes in computer science, vol. 3621; 2005. p. 17–36.
Wang Xiaoyun, Yu Hongbo. How to break MD5 and other hash functions. In: Cramer R, editor. EUROCRYPT 2005. LNCS, vol. 3494. International Association for Cryptologic Research 05; 2005. p. 19–35.
Xiaoyun W, et al. An attack on hash function HAVAL-128. Sci China Ser F Inf Sci 2005;48(5):1–12.

Please cite this article in press as: Venkateswara Rao P, et al., ASH-512: Design and implementation of cryptographic hash algorithm using co-ordinate geometry concepts, Journal of Information Security and Applications (2014), http://dx.doi.org/10.1016/j.jisa.2014.10.006