Journal of Computing and Information Technology - CIT 21, 2013, 4, 223–233
doi:10.2498/cit.1002181

Enhancing the Security Level of SHA-1 by Replacing the MD Paradigm

Harshvardhan Tiwari and Krishna Asawa
Jaypee Institute of Information Technology (JIIT), Noida (Uttar Pradesh), India

Cryptographic hash functions are important cryptographic techniques and are used widely in many cryptographic applications and protocols. All the MD4 design based hash functions such as MD5, SHA-0, SHA-1 and RIPEMD-160 are built on Merkle-Damgård iterative method. Recent differential and generic attacks against these popular hash functions have shown weaknesses of both specific hash functions and their underlying Merkle-Damgård construction. In this paper we propose a hash function which follows design principle of SHA-1 and is based on dither construction. Its compression function takes three inputs and generates a single output of 160-bit length. An extra input to a compression function is generated through a fast pseudo-random function. Dither construction shows strong resistance against major generic and other cryptanalytic attacks. The security of proposed hash function against generic attacks, differential attack, birthday attack and statistical attack was analyzed in detail. It is exhaustedly compared with SHA-1 because hash functions from SHA-2 and SHA-3 are of higher bit length and known to be more secure than SHA-1. It is shown that the proposed hash function has high sensitivity to an input message and is secure against different cryptanalytic attacks.

Keywords: cryptographic hash function, MD4, SHA-1, RIPEMD-160, generic attacks

1. Introduction

The rapid growth of new digital technologies increased the demand of information security in communication. Cryptographic hash functions are being widely used in different security applications and protocols such as digital signature, message authentication code, SSL, TLS, etc. for ensuring the integrity and authenticity of information. Cryptographic hash functions are functions that compress an input message of arbitrary length to an output with short fixed length, the hash value. Collision resistance, preimage resistance and second preimage resistance are three important security properties of a hash function. Collision resistance means it is computationally infeasible to find two distinct inputs X, X' with $H(X) = H(X')$. It is practically impossible to find the preimage X of $H(X)$, when $H(X)$ is given, this is referred to as preimage resistance. Finding $X \neq X'$ with $H(X) = H(X')$, when X and $H(X)$ are given, should also be infeasible. This property is called second preimage resistance. Due to the birthday paradox, an ideal hash function that generates an n-bit hash value requires evaluating about $2^{n/2}$ messages to find any pair of messages having the same hash value. Also, it requires 2^n hash computations for finding preimage and second preimage. There are three main categories of hash functions, namely hash functions based on block cipher, hash functions based on modular algorithm and dedicated hash functions [1, 2]. Most widely used hash functions are MD4 [3] design based dedicated hash functions. These hash functions use traditional Merkle-Damgård iterative structure [4, 5]. The input message $M$is padded to obtain a message of length multiple m-bits and divided into t blocks of equal length. The hash function H can then be described as follows:

$$h_0 = IV, \quad h_i = f(h_{i-1}, M_i), \quad 1 \leq i \leq t,$$
$$H(M) = h_t$$

f is a collision resistant compression function, h_i is the i^{th} chaining variable and IV is the initial value of the chaining variable. Recent attacks presented by many researchers have exposed flaws in both Merkle-Damgård construction and specific hash functions. Some attacks against Merkle-Damgård construction are fixed point [6], multicollision attack [7], second preimage attack [8], herding attack [9]. In this paper,

we propose a hash function DSHA-1 that takes an input message of arbitrary length and converts it into a 160-bit hash value. The compression function of the proposed hash function uses dithering design. Dither construction [10] is obtained by providing an additional input to the Merkle-Damgård construction. Its padding and splitting of message is similar to Merkle-Damgård construction. The iterative structure of this design provides good resistance against generic and other cryptanalytic attacks. The compression function of the proposed hash function is defined as follows:

$$h_0 = IV, h_i = f(h_{i-1}, M_i, d_i),\ 1 \leq i \leq t,$$
$$H(M) = h_t$$

where d_i is a dither input, obtained from a pseudo random function. In [11], this function is used as a replacement for Boolean functions of step operations in SHA-1. For each 512-bit message block M_i there is a different set of 80 32-bit dither values d_i.

The rest of this paper is organized as follows; Section 2 presents the related work. The proposed hash function is presented in Section 3. Section 4 contains the security analysis of DSHA-1. The performance analysis of hash function is presented in Section 5. The paper is concluded in Section 6.

2. Related work

A number of hash functions have been proposed, most of them have been influenced by the design of the MD4 hash function. The MD4 hash function was proposed by Rivest [3]. The algorithm produces hash values of 128 bits in length. It is a very fast hash function optimized for 32-bit architectures. Extended version of MD4 generates 256-bit hash value. MD4 pads a message by appending single bit 1 followed by variable number of 0's until the length of the message is 448 modulo 512 and then the 64-bit length of the message is appended as two 32-bit words. Other MD4 variants also use the same padding rule. The compression function of MD4 takes as input 128-bit chaining variable and a 512-bit message block and maps this to a new chaining variable. Each run of a compression function consists of three rounds and 48 sequential steps (each round consists of 16 steps), where each step is used to update the value of one of the four registers. Every round of MD4 compression function uses a different non-linear Boolean function. In [12], the first attack on MD4 was published. The attack was on the last two rounds of MD4. Authors showed that if the first round is omitted, then collision in MD4 can be found easily. Merkle showed an attack on the first two rounds of MD4, but this work was never published. Dobbertin [13] showed that MD4 is not a collision resistant hash function. He also showed that the first two rounds of MD4 are not one-way.

MD5 was also designed by Rivest as a strengthened version of MD4 [14]. It generates 128-bit hash value. Padding, parsing and processing of MD5 is similar to MD4, but some changes have been made to MD4. Changes include the addition of one extra round along with a new round function and redefined second round function. The compression function uses four rounds, each round has sixteen steps. Four non-linear Boolean functions, 64 different additive constants are used in MD5. Each message, like the MD4, after it has been appended by padding bits, is processed in blocks of 512 bits. In [15], authors found pseudo-collision for MD5. Dobbertin [16] published an attack that found a collision in MD5. At Crypto'04, collision in MD5, as well as collisions in other hash functions such as MD4, RIPEMD and HAVAL-128 was announced [17].

Other MD4 design based hash functions are from SHA-family. The original design of the hash function SHA-0 was designed by NSA and published by NIST as FIPS PUB 180 [18]. Two years later, SHA-0 was withdrawn due to a flaw found in it and replaced by SHA-1, published by NIST as FIPS PUB 180-1[19]. Both SHA-0 and SHA-1 produce a hash value of 160-bits. The only difference between these two versions is that SHA-1 uses a single bitwise rotation in its message schedule. Padding is done in the same way, then a 512-bit message block is split into sixteen 32-bit words and expanded into eighty 32-bit words using a message expansion relation. Each block is processed in 4 rounds consisting of 20 steps each. These four rounds are structurally similar to one another, with the only difference that each round uses a different Boolean function and one of four different additive constants. A complete round of SHA-0/1 is made up of 80 steps. NIST published

SHA-2 family as FIPS PUB 180-2 [20]. SHA-2 family consists of four hash functions, SHA-224, SHA-256, SHA-384, and SHA-512. SHA-224 and SHA-384 are the truncated versions of SHA-256 and SHA-512 respectively. The first result of cryptanalysis of SHA-0 was presented at Crypto'98. Chabaud and Joux [21] found collision with complexity 2^{61}. This was a differential attack and faster than generic birthday paradox attack. Biham and Chen [22] found two near-collisions of the full compression function of SHA-0. In [23], authors presented collision for the full SHA-0 and reduced SHA-1 algorithms. In [24], authors showed collision in SHA-0 in 2^{39} operations. Rijmen and Oswald [25] published an attack on a reduced version of SHA-1. In [26], authors presented collisions in full SHA-1 with less than 2^{69} hash operations.

The RIPEMD hash function was designed in the framework of the European Race Integrity Primitives Evaluation (RIPE) project. The design of RIPEMD is based on MD4; its compression function consists essentially of two parallel schemes of the MD4 compression function. It generates 128 bit message digest. Later, two strengthened versions of RIPEMD are released, RIPEMD-128 and RIPEMD-160. RIPEMD-128 also produces 128 bit message digest as its predecessor. The RIPEMD-160 hash function [27] processes 512-bit input message blocks and produces a 160-bit hash value. Both RIPEMD-128 and RIPEMD -160 are extended to RIPEMD-256 and RIPEMD-320 respectively.

In 2007 NIST introduced a public call for new cryptographic hash algorithms [28]. The intent of the competition was to identify modern secure hash functions and to define the new SHA-3 family. Keccak was selected as SHA-3 standard.

3. DSHA-1: A 160-bit hash function proposal

In this section, we describe the DSHA-1 hash function. The DSHA-1 hash function is based on the design principle of SHA-1. The DSHA-1 hash function produces a message digest of 160-bit length from a message of length less than 2^{64} bits. It follows a dither construction, based on a compression function that takes three inputs: chaining variables, message block and dither input to generate 160-bit hash value. Dither inputs are generated through a pseudorandom number generator. The initial working variables are specified constants, and the final chaining value is used as the output. The compression function processes one 512-bit message block per iteration. Message expansion is applied to each 512-bit message block, where 16 32-bit message words are expanded to 80 32-bit message words. The compression function consists of four rounds, each round is made up of a sequence of 20 steps. DSHA-1 uses five 32-bit chaining variables to which new values are assigned in each round.

The proposed algorithm includes two main stages for the computation of 160-bit hash value: first is preprocessing stage and second is computation stage. Preprocessing stage contains three steps: message padding, message parsing and initialization of eight chaining variables. The computation involves message expansion and application of compression function. The detailed specifications of hash function DSHA-1 is given below.

3.1. Message padding

The message M is padded before hash computation begins. The purpose of this padding is to ensure that the padded message is a multiple of 512. The message is always padded although the message already has the desired length. The input message is processed by 512-bit block. The hash function pads a message by appending a single bit 1 next to the end of a message, followed by zero or more bit 0's until the length of the message and finally 64-bit message length. Suppose that the length of the message M is l bits, append the bit 1 to the end of the message, followed by m zero bits, where m is the smallest, non-negative solution to the congruence $l + 1 + m \equiv 448 \mod 512$. Then represent the length of M (before padding) by a 64-bit block and append it to the padded message. The entire message length is now exact multiple of 512.

3.2. Parsing the message

After a message has been padded, it must be parsed into t 512-bit blocks, $M = M_1, M_2, ..., M_t$ before the hash computation can begin.

3.3. Setting the initial chaining variables

The 160-bit chaining variables are used to hold intermediate and final results of the hash function. The five chaining variables are initialized to the following hexadecimal values.

$$A = \text{0x67452301}$$
$$B = \text{0xEFCDAB89}$$
$$C = \text{0x98BADCFE}$$
$$D = \text{0x10325476}$$
$$E = \text{0xC3D2E1F0}$$

3.4. Constants

The four constants (in hexadecimal) are defined as:

$$K_t = \text{0x5A827999} \quad \text{for } t = 0, \ldots, 19$$
$$K_t = \text{0x6ED9EBA1} \quad \text{for } t = 20, \ldots, 39$$
$$K_t = \text{0x8F1BBCDC} \quad \text{for } t = 40, \ldots, 59$$
$$K_t = \text{0xCA62C1D6} \quad \text{for } t = 60, \ldots, 79$$

3.5. Boolean functions

The Boolean function F_t is defined as:

$$F_t = (B \wedge C) \vee (\neg B \wedge D) \quad \text{for } t = 0, \ldots, 19$$
$$F_t = (B \oplus C \oplus D) \quad \text{for } t = 20, \ldots, 39$$
$$F_t = (B \wedge C) \vee (B \wedge D) \vee (C \wedge D) \quad \text{for } t = 40, \ldots, 59$$
$$F_t = (B \oplus C \oplus D) \quad \text{for } t = 60, \ldots, 79$$

3.6. Dither input

We have taken a pseudo random function which generates a unique pseudo random number according to its input message word. Total of 80 32-bit pseudo-random numbers are generated. These random numbers are given to compression function as a third input. Each step operation makes the use of exactly one 32-bit dither input. The algorithm details are:

$$F(i) = \left\{ x_i * \sqrt{2} \right\}$$

where $\{.\}$ is a number's fractional part and x_i is the value defined by users. In this algorithm we have taken a 32-bit message word as x_i. The pseudo code of this algorithm is given below:

$temp = Math.sqrt(2) * x_i;$
$for\, i = 1\, to\, k\, \{$
$a[i] = tostring(temp - Math.floor(temp));$
$a[i] = substring(a[i], 2, 11); \}$

Different dither inputs for compression function are computed as:

$$d_i = F_i \left\{ W_i * \sqrt{2} \right\}, \quad \text{for } i = 0, \ldots, 79.$$

The benefits of this algorithm are high speed of generating and the longest repeating period.

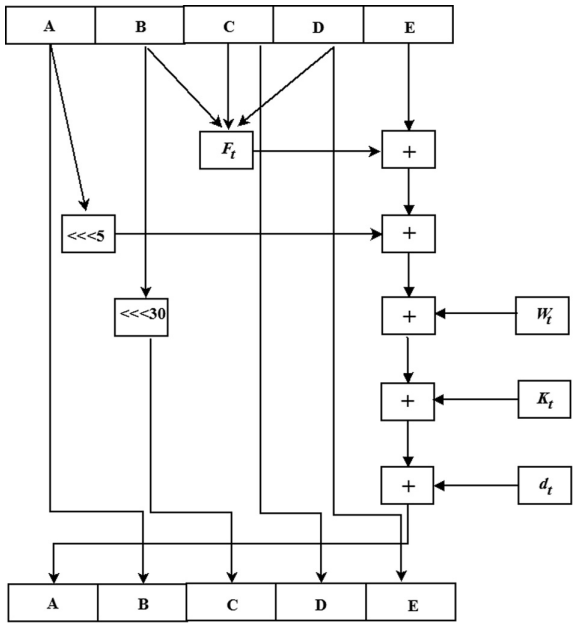

Figure 1. Step operation of DSHA-1.

3.7. Message expansion

The compression function processes one 512-bit message block per iteration. The 512-bit message block is divided into sixteen 32-bit words $W{=}W_0, W_1, ..., W_{15}$. These 16 32-bit words are expanded to 80 32-bit words with the help of the following relation, for $t = 16, ..., 79$

$$W = (W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) <<< 1$$

3.8. Processing

Execution of the compression function involves 80 steps. The compression function is composed of four rounds of processing where each round is made up of twenty steps. The processing of a compression function is defined as follows:

(a) Set $\bar{A} = A, \bar{B} = B, \bar{C} = C, \bar{D} = D, \bar{E} = E$.

(b) for $t = 16, \ldots, 79$

$$T = (A<<<5)+F_t(B,C,D)+E+W_t+K_t+d_t$$
$$E = D$$
$$D = C$$
$$C = B<<<30$$
$$B = A$$
$$A = T$$

3.9. Output

Upon the completion of the compression function the output is obtained as:

$$A = \bar{A} + A$$
$$B = \bar{B} + B$$
$$C = \bar{C} + C$$
$$D = \bar{D} + D$$
$$E = \bar{E} + E$$

After processing the last 512-bit message block, we get final 160-bit hash value.

4. Security analysis

Cryptographic hash function must satisfy three properties: Preimage resistance, second preimage resistance and collision resistance. In preimage attack on an n-bit hash function, for a given hash value $h(M)$an attacker tries to find a message M. An attack against second preimage resistance, given a message M and $h(M)$ finds another message M' such that $h(M) = h(M')$. Both brute force preimage or second preimage attacks require 2^n computation of hash values. DSHA-1 produces a 160 bit output, so finding preimage and second preimage in DSHA-1 requires 2^{160} operations. Collision resistance means an attacker should not be able to find two messages $M \neq M'$ with $h(M) = h(M')$. For a hash function with n-bit hash value, an adversary will have to try at least $2^{n/2}$ operations to find two different messages with the same hash value. For a 160 bit hash function DSHA-1, the complexity of collision attack is 2^{80}. In this section, our aim is to quantify the security of DSHA-1 against major cryptanalytic attacks. In order to accomplish this task, the following analysis has to be considered.

4.1. Length extension attack

For a given hash function h, if one can find a collision for two messages M and M' with $M \neq M'$ such that $h(M)$ and $h(M')$ collide, one can apply a length extension attack. For any message m, one can easily produce a collision for $(M \parallel m)$ and $(M' \parallel m)$ as $h(M \parallel m) = h(M' \parallel m)$. Padding rule of the algorithm avoids such type of attacks since we concatenate the length of the message to the message itself. Another attack can be as follows: for a known $h(M)$ one can compute the hash value $h(M \parallel P \parallel m)$ for any suffix m, if the length of an unknown message M is known as well as padding P of M. DSHA-1 prevents this kind of attack because each step uses a different and unique pseudorandom number which restricts an attacker from recovering the internal steps and gaining any information of prior step operations.

4.2. Multi-collision attack

Joux found that when iterative hash functions are used, finding a set of 2^k messages all colliding on the same hash value is as easy as finding k single collision for the hash function. Finding a collision, in the compression function, i.e., a single block collision, one can find k of such collisions, each starting from the chaining value produced by the previous block collision. In other words, one has to find two message blocks M_i and M_i' where $M_i \neq M_i'$ with $f(H_{i-1}, M_i) = f(H_{i-1}, M_i')$ where f represents the compression function and H_i the chaining value. Then it is possible to construct 2^k messages with the same hash value by choosing for block i either the message block M_i or M_i'. Joux showed that the concatenation of two different hash functions is not more secure against collision attacks than the strongest one. The complexity of the attack depends on the size of internal state of compression function since internal state of compression function of DSHA-1 is small and not big enough to prevent this attack.

4.3. Herding attack

The herding attack works as follows: an attacker takes 2^k chaining values which are fixed or randomly chosen. Then he chooses $2^{n/2-k/2}$ message blocks. He computes the output of the compression function for each chaining value and each block. It is expected that for each chaining value there exists another chaining value, such that both collide to the same value. The attacker stores the message block that leads to such a collision in a table and repeats this process again with the newly found chaining values. Once the attacker has only one chaining value, it is used to compute the hash value to be published. To find a message whose chaining value is among the 2^k original values, the attacker has to perform 2^{n-k} operations. For such a message the attacker can retrieve from the stored messages the message blocks that would lead to the desired hash value. DSHA-1 uses a 32-bit pseudo random sequence in each step. It is difficult to build collision using a diamond structure with fixed initial values, random message and a random dither sequence.

4.4. Fixed point attack

Dean found that fixed points in the compression function can be used for a second preimage attack against long messages. Kelsey and Schneier extend this result and provide an attack to find a second preimage on a Merkle-Damgård construction with Merkle-Damgård strengthening much faster than 2^n efforts. The complexity of the attack is determined by the complexity of finding expandable messages. These are messages of varying sizes such that all these messages collide internally for a given IV. Fixed point attacks in this form cannot be applied to the DSHA-1 because we include the random dither value in each iteration of compression function which does not allow finding expandable messages.

4.5. Slide attack

Slide attacks are common in block cipher cryptanalysis, but they are also applicable to hash functions. Given a hash function h and two messages M and M' where M is a prefix of M', one can find a slid pair of messages M and M' such that the last message input block of the longer message M' performs only an additional blank round, e.g. for sponge constructions. These two messages are then slid by one blank round. This attack allows to recover the internal state of a slid pair of messages. Involving a different dither value in each step does not allow finding slid pairs of messages and avoids the possibility of slide attacks.

4.6. Differential attack

The essential idea of differential attack on hash functions, as used to break MD5 and SHA-0/1, is to exploit a high-probability input/output differential over some component of the hash function, e.g. under the form of a perturb-and-correct strategy for the latter functions, exploiting high probability linear/non-linear characteristics. The method finds the collision for two identical messages M and M'. Perturbation refers to the initial change of a bit in M'. Correction refers to those bits in the message M'which need to be changed to make the variables of both messages equal in subsequent rounds. The random dither values in the computation steps of compression function provide the high diffusion and strengthen the function against differential path. The correction methods fail to correct the differences at the end of all computation rounds. Number of rounds increases difference between the messages. It is very hard to control differences for DSHA-1, thus eliminating the possibility of differential attack.

4.7. Meet-in-the-middle attack

This attack is a variation of birthday attack and is applicable to the hash functions that use a round function. Instead of message digest, intermediate chaining variables are compared. This attack enables a cryptanalyst to construct a message with a pre-specified message digest, which is not possible in case of a simple birthday attack. The attacker generates s_1 samples for the first part and s_2 samples for the last part of a forged message. The attacker then goes forwards from initial value and goes backwards from the hash value and expects that the two intermediate values collide to the same value.

Onewayness of the compression function restricts an attacker from constructing a message with a pre-specified hash value.

4.8. Correcting block attack

In this attack, the cryptanalyst uses a pair of existing messages and its corresponding hash value and tries to change one or more message blocks such that the resulting hash value remains unchanged. A correction block attack is used to produce a collision. Starting with two arbitrary messages M and M' and appending one or more correcting blocks denoted by such that the extended messages have the same hash values. Computation steps of DSHA-1 are redundant enough to resist the correcting block attack.

5. Performance analysis

5.1. Randomness

We have taken an input message M of 512 bit length and computed corresponding hash value. By changing the i^{th} bit of M, new modified messages M_i have been generated, for $1 \leq i \leq 512$. Then we generated hash values of all these new messages and finally computed Hamming distances between hash values of original message and modified messages. Ideally, it should be 80. But we found that these Hamming distances were between 68 and 97 for above messages. Range of distances is given in Table 1. The average Hamming distance is 81.61. Distribution of distances is shown in Figure 2.

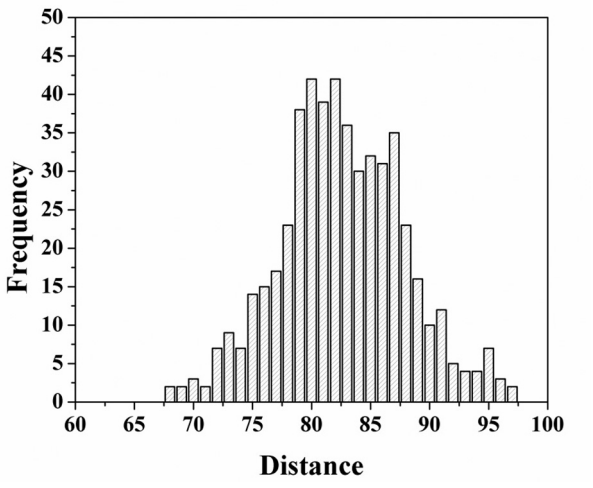

Figure 2. Frequency distribution of distances for DSHA-1.

Distances	Hash pairs	Percentage (%)
80 ± 5	316	61.71
80 ± 10	467	91.22
80 ± 15	505	98.63

Table 1. Range of distances for DSHA-1.

Distances	Hash pairs	Percentage (%)
80 ± 5	281	54.88
80 ± 10	432	84.37
80 ± 15	487	95.11

Table 2. Range of distances for SHA-1.

5.2. Bit variance test

The bit variance test consists of measuring the impact on the digest bits by changing input message bits. Bits of an input message are changed and the corresponding message digests (for each changed input) are calculated. Finally, from all the digests produced, the probability P_i for each digest bit to take on the value of 1 and 0 is measured. If $P_i(1) = P_i(0) = 1/2$ for all digest bits i $1 \leq i \leq n$, where n is the digest length, then the hash function under consideration has attained maximum performance in terms of the bit variance test. Therefore, the bit variance test actually measures the uniformity of each bit of the digest. Since it is computationally difficult to consider all input message bit changes, we have evaluated the results for only up to 513 files and found the results shown in Table 3.

Hash function	Mean frequency of 1s(Expected)	Mean frequency of 1s(Calculated)
SHA-1	256.50	249.11
DSHA-1	256.50	256.67

Table 3. Bit variance analysis.

The above analysis shows that hash function exhibits a reasonably good avalanche effect. Thus it can be used for cryptographic applications.

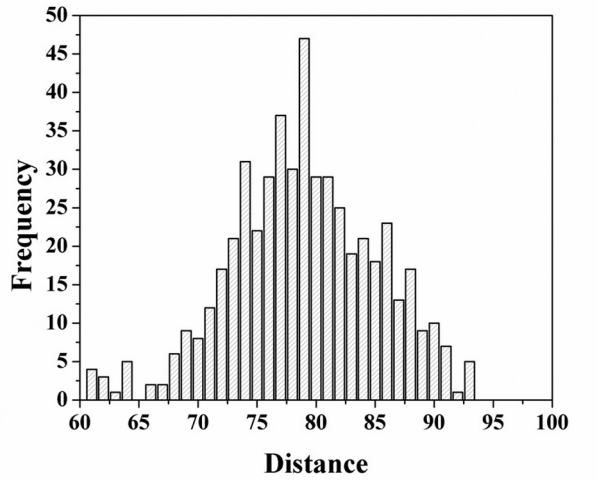

Figure 3. Frequency distribution of distances for SHA-1.

5.3. Statistical analysis of confusion and diffusion

Confusion and diffusion are two basic design criteria for encryption algorithms as well as hash functions. Diffusion means to spread the influence of a single plaintext or key bits over as much of the ciphertext as possible so as to hide the statistical structure of the plaintext. Confusion means to exploit some transformations to hide any relationship between the plaintext, the ciphertext and the key, thus making cryptanalysis more difficult. For the hash value in binary format, each bit is only 1 or 0. So the ideal diffusion effect should be that any tiny changes in the input lead to the 50% changing probability of each output bit. We have performed the following diffusion and confusion test [29]: A message is randomly chosen and hash value is generated. Then a bit in the message is randomly selected and toggled and a new hash value is generated. Finally, we compare two hash values and count number of changed bit as B_i. The experiment is carried out N times. Usually, four statistics are defined as follows:

Mean changed bit number:

$$\bar{B} = \frac{1}{N}\sum_{i=1}^{N} B_i$$

Mean changed probability:

$$P = (\bar{B}/160) \times 100\%$$

Standard deviation of the changed bit number:

$$\Delta B = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(B_i - \bar{B})^2}$$

Standard deviation:

$$\Delta P = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(B_i/160 - P)^2} \times 100\%$$

where N is the total number of test and B_i is the number of changed bits in the i^{th} test. Distribution of changed bit number is shown as Figure 4, where $N = 512$.

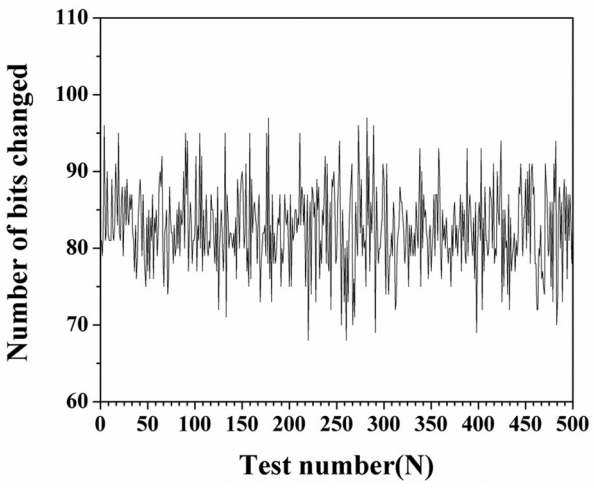

Figure 4. Distribution of changed bit number for DSHA-1.

Through the tests with $N = 64, 128, 256, 512$, respectively, the corresponding data are listed in Table 4.

N	$\bar{B}$	$P\%$	ΔB	$\Delta P\%$
64	80.4609	50.2881	6.3387	3.9617
128	80.6035	50.3772	6.2059	3.8787
256	80.5215	50.3259	6.1935	3.8709
512	80.4171	50.2606	6.1707	3.8567

Table 4. Statistics of number of changed bits for DSHA-1.

N	$\bar{B}$	$P\%$	ΔB	$\Delta P\%$
64	79.4519	49.4289	6.6232	3.5634
128	79.6023	49.3874	6.4522	3.2356
256	80.0313	49.1093	6.6743	3.8431
512	79.3926	49.3945	6.3472	3.6452

Table 5. Statistics of number of changed bits for SHA-1.

Based on the analysis of the data in Table 4, we can draw the conclusion: the mean changed bit number $\bar{B}$ and the mean changed probability P are both very close to the ideal value 80 bit and 50%. ΔB and ΔP are very little, which indicates that the capability for diffusion and confusion is very stable.

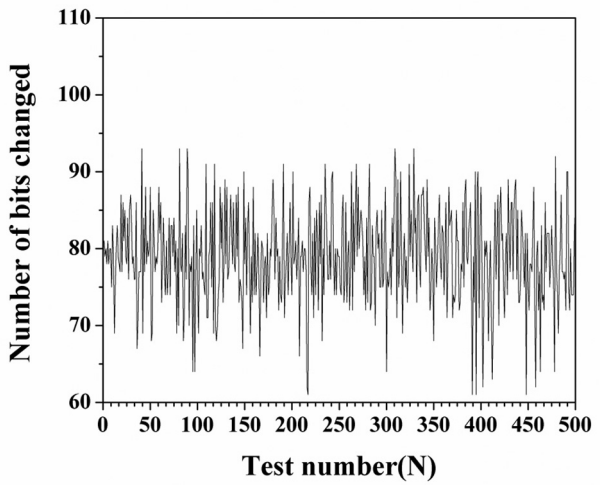

Figure 5. Distribution of changed bit number for SHA-1.

5.4. Analysis of collision resistance

Collision means that the hash results are identical to different random inputs. In order to investigate the collision resistance capability of the proposed hash function, we have performed two collision tests. In the first experiment, the hash value for a randomly chosen message is generated and stored in ASCII format. Then a bit in the message is selected randomly and toggled and thus a new hash value is generated and stored in the same format. Two hash values are compared and the number of characters in this format with the same value at the same location in hash value is counted. The absolute difference of the two hash results is calculated by using the following formula:

$$AD = \sum_{i=1}^{N} \left| dec(e_i) - dec(e_i') \right|$$

where e_i and e_i' are the i^{th} ASCII character of the original and the new hash value, respectively, $dec()$ converts the entries to their equivalent decimal values. This kind of collision test is performed 2048 times. The maximum, minimum and mean values of AD are listed in Table 6.

AD	Max	Min	Mean	Mean/char
Values	2690	993	1702.6372	85.13
Values	2332	695	1642.63	83.64

Table 6. Absolute difference of DSHA-1 and SHA-1.

In the second experiment, the hash value for a randomly chosen message is generated and stored in ASCII format similarly. What is focused in this experiment is the possibility of colliding between every two hash results, thus every two hash results should be compared. The simulation is performed 2048 times. The plot of the distribution of the number of ASCII characters with the same value at the same location is given in Figure 6. The maximum number of equal entries in the Figure 6 is 2. So the hash results could resist collision well from the Figure 6.

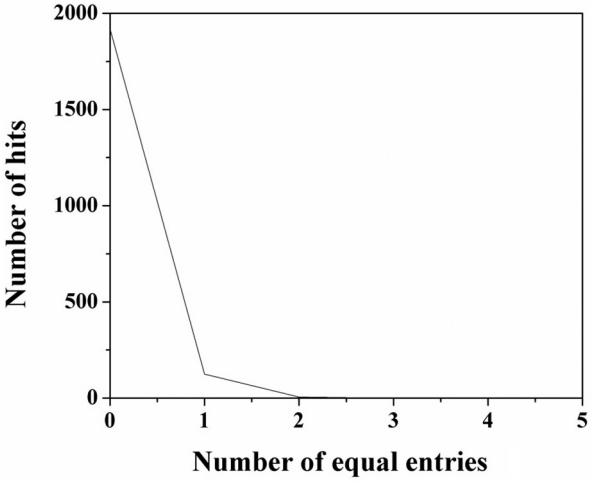

Figure 6. Distribution of the number of ASCII characters with the same value at the same location in the hash value for DSHA-1.

5.5. Robustness against differential cryptanalysis

We studied the robustness of the proposed hash function against differential cryptanalysis. This attack analyzes the plaintext pairs along with their corresponding pairs of hashes. For example, if the difference between 2 messages is 2 bits, (i.e., say, $d = 2$) then the message digest

pair difference d' for the corresponding 2 message digests can be calculated. From the distribution of d' corresponding to different message pairs, the standard deviation (σ) is calculated. If $\sigma < 10\%$, then the hash function is secure against differential cryptanalysis. For the experiment, input message of 10 bytes was considered. The experiments were run for all possible $d = \{1, 2, 4, 8, 16, 32\}$ bit differences for an input message. The results in Table 7 show that the proposed hash function is secure against the differential attack.

d	1	2	4	8	16	32
σ (DSHA-1)	6.18	6.11	6.07	6.14	6.23	6.09
σ (SHA-1)	7.56	8.34	10.02	8.87	10.14	7.19

Table 7. Results for differential cryptanalysis.

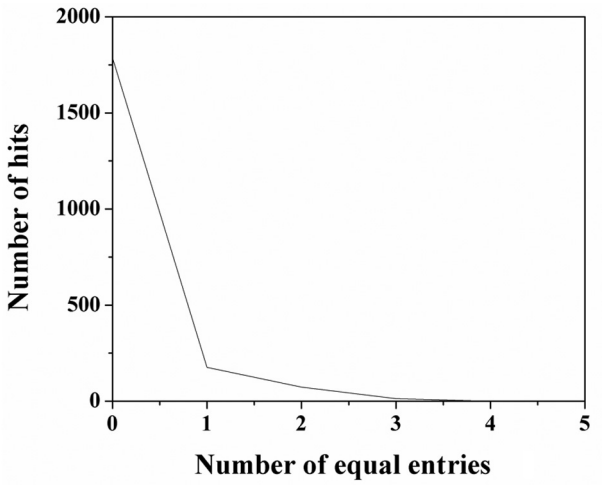

Figure 7. Distribution of the number of ASCII characters with the same value at the same location in the hash value for SHA-1.

6. Conclusion

The proposed hash function processes a message of arbitrary length by 512-bit blocks and produces as output a 160-bit hash value or message digest. DSHA-1 is built on dither construction. The compression function of DSHA-1 takes three input parameters: 512-bit message block, 160-bit chaining variable and 80 32-bit words of dither input, and produces a single output of 160-bit length. Dither input words are generated through a pseudo-random function. This function uses message sub-blocks as a key. Random dither input words to a compression function provide larger minimum distance between similar words, high randomness, good mixing of bits and lesser control over the propagation of difference in the words. The iterative structure of a proposed hash function preserves all the three security properties: collision resistance, preimage and second preimage and achieves the high level security against major generic attacks. We have analyzed the proposed hash function for its randomness and security. The bit variance test has been performed for one bit changes. The result of bit variance test shows that DSHA-1 achieves a good degree of avalanche effect, i.e. when a single input bit is complemented, each of the output bits changed with a probability of 0.5. Thus proposed hash function pass the bit variance test. The statistical analysis of DSHA-1 indicates that it has strong and stable confusion and diffusion capability. The calculated mean changed bit number and mean changed probability are 81.50 and 50.31% respectively, both very close to the ideal value 80 bit and 50% while standard deviation of the changed bit number and standard deviation are very little, which indicates the capability for confusion and diffusion is very stable. It possesses high message sensitivity and good statistical properties.

References

[1] B. SCHNEIER, *Applied cryptography*. John Wiley & Sons, 1996.

[2] A. J. MENEZES, P. C. VAN OORSCHOT, S. A. VANSTONE, *Handbook of applied cryptography*. CRC Press, 1997.

[3] R. RIVEST, The MD4 message digest algorithm. *CRYPTO'90*, LNCS, **537** (1991), 303–311.

[4] R. MERKLE, One way hash functions and DES. *CRYPTO' 89*, LNCS, **435** (1990), 428–446.

[5] I. B. DAMGÅRD, A design principle for hash functions. *CRYPTO' 89*, LNCS, **435** (1990), 416–427.

[6] R. D. DEAN, Formal aspects of mobile code security. PhD Thesis, Princeton University, 1999.

[7] A. JOUX, Multicollisions in iterated hash functions. *CRYPTO'04*, LNCS, **3152** (2004), 306–316.

[8] J. KELSEY, B. SCHNEIER, Second preimages on n-bit hash functions for much less than 2^n work. *EUROCRYPT'05*, LNCS, **3494** (2005), 474–490.

[9] J. KELSEY, T. KOHNO, Herding hash functions and the Nostradamus attack. *EUROCRYPT' 06*, LNCS, **4004** (2006), 183–200.

[10] R. RIVEST, Abelian square-free dithering for iterated hash functions. *ECRYPT Hash Function Workshop*, 2005.

[11] A. ABROR, S. LEE, Y. LEE, Modified SHA-1 hash function (mSHA-1). *ITC-CSCC' 09*, (2009), pp. 1320–1323.

[12] B. DEN BOER, A. BOSSELAERS, An attack on the last two rounds of MD4. *CRYPTO '91*, LNCS, **576** (1992), 194–203.

[13] H. DOBBERTIN, Cryptanalysis of MD4. *FSE' 96*, LNCS, **1039** (1996), 53–69.

[14] R. RIVEST, The MD5 message digest algorithm. Request for Comments (RFC) 1321, Internet Engineering Task Force, 1992.

[15] B. DEN BOER, A. BOSSELAERS, Collisions for the compression function of MD5. *EUROCRYPT '93*, LNCS, **765** (1994), 293–304.

[16] H. DOBBERTIN, Cryptanalysis of MD5. *EUROCRYPT'96*, 1996.

[17] X. WANG, X. D. FENG, X. LAI, H. YU, Collisions for hash functions MD4, MD5, HAVAL-128 and RIPEMD. *CRYPTO' 04*, 2004.

[18] NIST, Secure hash standard (SHS), Federal Information Processing Standards 180, 1993.

[19] NIST, Secure hash standard (SHS), Federal Information Processing Standards 180-1, 1995.

[20] NIST, Secure hash standard (SHS), Federal Information Processing Standards 180-2, 2002.

[21] F. CHABAUD, A. JOUX, Differential collisions in SHA-0. *CRYPTO'98*, LNCS, **1462** (1998), 56–71.

[22] E. BIHAM, R. CHEN, Near-collisions of SHA-0. *CRYPTO'04*, LNCS, **3152** (2004), 290–305.

[23] E. BIHAM, R. CHEN, A. JOUX, P. CARRIBAULT, C. LEMUET, W. JALBY, Collision of SHA-0 and reduced SHA-1. *EUROCRYPT'05*, LNCS, **3494** (2005), 36–57.

[24] X. WANG, H. YU, Y. L. YIN, Efficient collision search attacks on SHA-0. *CRYPTO' 05*, LNCS, **3621** (2005), 1–16.

[25] V. RIJMEN, E. OSWALD, Update on SHA-1. *RSA'05*, LNCS, **3376** (2005), 58–71.

[26] X. WANG, Y. L. YIN, H. YU, Finding collisions in the full SHA-1. *CRYPTO' 05*, LNCS, **3621** (2005), 17–36.

[27] H. DOBBERTIN, A. BOSSELAERS, B. PRENEEL, RIPEMD-160– A strengthened version of RIPEMD. *FSE'96*, LNCS, **1039** (1996), 71–82.

[28] B. PRENEEL, The NIST SHA-3 Competition: A perspective on the final year. *AFRICACRYPT'11*, LNCS, **6737** (2011), 383-386.

[29] K. W. WONG, A combined chaotic cryptographic and hashing scheme. *Physics letters A*, **307**(5-6) (2003), 292–298.

Received: July, 2013
Revised: November, 2013
Accepted: November, 2013

Contact addresses:

Harshvardhan Tiwari
Jaypee Institute of Information Technology (JIIT)
Noida (Uttar Pradesh)
India
e-mail: tiwari.harshvardhan@gmail.com

Krishna Asawa
Jaypee Institute of Information Technology (JIIT)
Noida (Uttar Pradesh)
India

HARSHVARDHAN TIWARI received his B.E. and M.Tech. degrees in Computer Science and Engineering from RGPV University, India, in 2005 and 2009, respectively. Currently he is a PhD scholar at JIIT University. His research interests are computer network, algorithms, network security and information security.

KRISHNA ASAWA is presently working with Jaypee Institute of Information Technology (JIIT), Noida (Uttar Pradesh), India, in the capacity of an Associate Professor. In 2002 Dr. Krishna received a PhD degree in Computer Science from Banasthali Vidyapith, Deemed to be University, Banasthali (Rajasthan), India. She completed her post graduate study (1993 in Computer Science) and her graduate study (1996 in Electronics) at the Faculty of Engineering, Jai Narain Vyas University Jodhpur (Rajasthan), India. Her area of interest and expertise includes soft computation, information security, knowledge and data engineering, multi agent communication and its modelling, machine cognition and its applications.