Design of Cryptographic Hash Functions based on MD and MD Variant

Synopsis of the Thesis submitted in fulfillment of the requirements for the Degree of

DOCTOR OF PHILOSOPHY

By

HARSHVARDHAN TIWARI **(**09403024)

Department of Computer Science and Engineering

JAYPEE INSTITUE OF INFORMATION TECHNOLOGY
(Declared Deemed to be University under section 3 of UGC Act)
A-10, SECTOR-62, NOIDA, INDIA
April 2014

1. INTRODUCTION

Hash functions have a long history in computer science. Their earliest application was that of mapping a large sparsely filled file into a much smaller one. In cryptography they are a fundamental building block for many security applications like integrity protection, message authentication, digital signature schemes, password storage and protection, confirmation of commitment, pseudo-random string generation and key derivation. A cryptographic hash function takes a message of arbitrary length and produces an output of fixed length. The output of a hash function is called hash value, message digest, or fingerprint. A cryptographic hash function should behave as much as possible like a random function while still being deterministic and computationally efficient. Cryptographic hash functions have to satisfy requirements of onewayness and collision resistance. Onewayness means that the method to calculate a hash value from a given message is easy, but it is computationally infeasible to generate any message that yields a given hash value. Collision resistance means it is extremely difficult to find two messages that have the same hash value. Cryptographic hash functions are classified into unkeyed hash functions and keyed hash functions. Unkeyed hash functions, also known as modification detection codes (MDCs), use message as a single input whereas keyed hash functions, also known as message authentication codes (MACs), can be viewed as hash functions which take two functionally distinct inputs, a message of arbitrary finite length and a fixed length secret key. In this thesis, the unkeyed hash functions are discussed and they are simply called hash functions. Formally, a hash function can be shown as follows: $h:\{0,1\}^* \rightarrow \{0,1\}^n$. As we can see from above, the input is arbitrary length of any binary string, and the output is n bits of binary string. We usually call n as the size of hash function. Preimage, second preimage and collision resistance are ground properties of a hash function. These are NIST core requirements for a cryptographic hash algorithm and are the requirements which are generally of most practical importance [1, 2]. Preimage resistance is important in some authentication scenarios and password storage where one does not send plain messages with their hash values, so if adversary can reverse the hash function he/she will be able to find the original message. Second preimage is for preventing the adversary from changing the original message in a way that the hash value remains unchanged. Collision resistance is stronger notion than preimage and second preimage resistance. Collision resistance always implies property second preimage resistance but does not imply preimage resistance. Collision resistance is easy to breach, so most cryptanalysis target collision attack. Collision resistance is important for digital signatures. The properties of

second preimage resistance and collision resistance may seem similar but the difference is that in the case of second preimage resistance, the attacker is given a message to start with, but for collision resistance no message is given; it is simply up to the attacker to find any two messages that yield the same hash value. The term 'computationally infeasible' or 'computationally difficult' means that the complexity of an algorithm to break any of these properties is not less than that of the generic attack required to break that property. For an n-bit hash function, we have a generic collision attack with complexity $2^{n/2}$, while brute force preimage or second preimage attacks have complexity 2^n. In case of collision attack, birthday attack is popularly used exhaustive search. The term 'computational easiness' might mean polynomial time and space; or more practically, within a certain number of machine operations or time units. Unkeyed hash function is further classified into oneway hash function (OWHF) and collision resistant hash function (CRHF) [3]. The construction of CRHF is hard than OWHF. CRHF usually deals with longer length hash values. Keyed hash functions are preferred to be used in authentication schemes and to verify data integrity whereas unkeyed hash function deals only with data integrity. MAC [4] does not provide non-repudiation.

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. Other approaches for building hash functions are chaos-based hash functions [28, 29] and cellular automata-based hash functions [30]. Among these, most widely used hash functions are MD4 [5] designed based dedicated hash functions. Security of MD4 design based hash functions have been damaged by recent dedicated attacks against internal structure of these hash functions and generic attacks against Merkle-Damgård construction [20]. This necessitates the transition to newer more secure hash functions that replace MD5 [6] and SHA-1 [8].

In this work we present two new hash function proposals MDA-192 and DSHA-1. Both hash functions are based on the design principle of SHA-1. MDA-192 encodes a message into a 192-bit hash value whereas DSHA-1 generates 160-bit output. In both the algorithms the input message is processed in 512-bit message blocks through repeated application of compression function.

MDA-192 is based on Merkle-Damgård transformation. Message expansion, data dependent rotations and the increased use of input message word in the step operation are the outstanding features of the proposed hash function. Message expansion of the algorithm expands 16 32-bit words to 96 32-bit words. The main motive of such expansion mechanism is to provide higher minimum distance between similar words, high randomness, good

mixing of bits and lesser control over the propagation of difference in the words. By using message words heavily in variable bit rotations and computation of step operations, we introduce the redundancy in the round functions of MDA-192. The algorithm is more secure and complicated for attacks.

DSHA-1 incorporates the idea of dither construction [21]. The compression function of DSHA-1 takes three inputs. 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.

Finally, a hash function design MNF-256 is suggested. MNF-256 is designed using the dither construction. Dither construction overcomes intrinsic limitations of the Merkle-Damgård approach. It provides strong resistance to multi-collision attacks [24], long second preimage attacks [25], and herding attack [26]. MNF-256 based on the principle of NewFORK-256 [22] (NFORK in this literature). It follows parallel structure i.e. its compression function uses three parallel branches to update chaining variables. It takes 512-bit message blocks and generates 256-bit hash value. A random sequence is added as an additional input to the compression function of MNF-256. The main goal of work is to enhance the security of SHA-1 and NFORK against recent proposed attacks.

2. LITERATURE REVIEW

2.1 Merkle-Damgård Construction and its weaknesses

From the early beginning of hash functions in cryptography, designers relied on the Merkle-Damgård (abbreviated to MD) construction. The MD construction was discovered by Merkle [27] and Damgård [20] in 1989 independently. Majority of famous hash functions such as MD4 [5], MD5 [6], SHA-0 [7], SHA-1[8], RIPEMD-160 [10] etc. follow the iterative MD method. A compression function which takes a fixed input length value and outputs a fixed length hash value is core component of this construction. One of its distinctive features is that it promotes the collision resistance and preimage resistance of the compression function to the full hash function: for instance, a collision on the compression function can be deduced efficiently from a collision on the full hash function. The inclusion of the message length at the end of the message is vital for preventing a number of attacks, including long-message attacks. Merkle-Damgård construction proves that the security of hash function relies on the security of the compression function. Thus, in order to build a collision resistant hash function, it is sufficient to design a collision resistant compression function. Recent results

[21] [24] [25] [26], however, highlight some intrinsic limitations of the MD approach. This includes being vulnerable to multi-collision attacks, long second pre-images attacks, and herding. Due to these structural weaknesses, researchers have proposed several variants of Merkle-Damgård construction [21] [31] [32].

2.2 Dedicated hash functions

Dedicated hash algorithms are specially designed from the scratch for the purpose of hashing a plain text with optimized performance and without being constrained to reusing existing system components such as block ciphers and modular arithmetic. These hash functions are not based on hard problems such as factorization and discrete logarithms. The most popular method of designing compression functions of dedicated hash functions is a serial successive iteration of a small step function. MD4 design based hash functions are the examples of dedicated hash functions. The MD4 was proposed by R.Rivest in 1990. Most commonly used hash functions are based on the design principles of MD4. MD5 was also proposed by R.Rivest in 1992 as a strengthen version of MD4. Both MD4 and MD5 produce 128-bit message digest. MD5 is slightly slower than MD4. The design principles of MD4 are used in SHA family. SHA-0 was developed in 1993 by National Security Agency as the Secure Hash Standard and SHA-1 was introduced in 1995 as a revision of SHA-0. SHA-1 was issued by NIST as FIPS PUB 180-1. Both SHA-0 and SHA-1 produce a message digest of 160-bit. NIST introduced new hash function standard FIPS PUB 180-2 in 2002 [9]. Three new hash functions, SHA-256, SHA-384 and SHA-512, collectively known as SHA-2, have been specified in this standard. Later on another hash function SHA-224 was added to this standard. Another popular hash function family is RIPEMD family. The RIPEMD family of hash functions was designed by combining sequential method and parallel structure. This method of designing is still reliable due to no effective attacks so far, except elementary versions of RIPEMD. The first RIPEMD hash function was introduced in 1992 under the European RIPE (RACE Integrity Primitives Evaluation) project. It produces 128-bit hash value. RIPEMD runs two almost identical copies of MD4 in parallel. Later two strengthen versions of RIPEMD are released, RIPEMD-128 and RIPEMD-160 [10]. RIPEMD-128 also produces 128- bit message digest as its predecessor. Both RIPEMD-128 and RIPEMD -160 are extended to RIPEMD-256 and RIPEMD-320 respectively. In 1998, MD4 was completely broken by Dobbertin [11]. In 2004, a team of researchers led by Xiaoyun Wang, announced collisions in MD5 as well as collisions in other hash functions including MD4, RIPEMD, and HAVAL-128 [12]. In 2005, they presented attacks against SHA-0 and SHA-1[13]. FORK

family hash functions can be viewed as the further extension of RIPEMD family. FORK-256 was the first hash function in FORK family, introduced in the first NIST hash workshop and at FSE 2006. Matusiewicz, Contini, and Pieprzyk attacked FORK-256 by using the fact that the functions f and g in the step operation were not bijective They used microcollisions to find collisions of 2-branch FORK-256 and collisions of full FORK-256 with complexity of $2^{126.6}$ [14]. Independently, Mendel, Lano, and Preneel [15] published the collision-finding attack on 2-branch FORK-256 using microcollisions and raised possibility of its expansion. At FSE 2007, Matusiewicz, Peyrin, Billet, Contini, Pieprzyk another attack which finds a collision with complexity of 2^{108}. FORK-256 was optimized by Danda [16]. NewFORK-256 hash function was introduced in 2007. It includes bijective function in step operation. Markku -Juhani O. Saarinen presented collision attack against NewFORK-256 (NFORK in this literature) using meet-in-the-middle technique [17]. For this he used a method for finding messages that hash into a significantly smaller subset of possible hash values. The complexity of this collision attack is $2^{112.9}$. This attack is also applicable for FORK-256.

In 2007 NIST introduced a public call for new cryptographic hash algorithms. The intent of the competition is to identify modern secure hash functions and to define the new SHA-3 family [18, 19]. NIST announced Keccak [23] as the winner of the competition.

2.3 Applications

Digital signatures are very important in information security. A digital signature authenticates electronic documents in a similar manner a handwritten signature authenticates printed documents. Digital signatures enable the authentication and non-repudiation of digital messages, assuring the recipient of a digital message of both the identity of the sender and the integrity of the message. Signature generation makes use of a private key to generate a digital signature. Signature verification makes use of a public key, which corresponds to, but is not the same as, the private key. Hash functions are used in conjunction with digital signature schemes, where a message is hashed first, and then the hash value, as a representative of the message, is signed in place of the original message. The digital signature is sent to the intended verifier along with the message. The verifier computes the hash value over the received message and verifies the signature by using the sender's public key. The security of digital signatures depends on the cryptographic strength of the underlying hash functions.

MACs Message integrity and authenticity are essential in security-related communications. Here, a recipient is expected to be able to verify that a received message, originally transmitted by a valid source, was not changed. Technically, verifying message integrity and

authenticity is based on the recipient's ability to prove to itself that the sender stores a valid secret key that was used when the message was transmitted. MAC is closely related to cryptographic hash functions, which play a fundamental role in many areas of modern cryptography. In software, these hash functions have throughput as much as one order of magnitude higher than DES. Several factors motivated their adoption as the basis for MAC algorithms: the additional implementation and deployment effort required to adopt these as MAC is minimal; MAC based on these outperform most other available options; and such MAC, avoiding the use of encryption algorithms, may have preferential export status. Consequently, MAC constructions based on these hash functions were adopted in Kerberos, SNMP, and SSL, and gained favour in the IPsec working group of the IETF. $MAC(M, K)$ is a oneway transformation of the message M and a secret key K shared with the verifier. The values M and $MAC(M, K)$ are both sent to the verifier. Upon receiving these values, the verifier generates himself a value $MAC'(M, K)$ based on the received M and the value of K known to him. If $MAC(M, K) = MAC'(M, K)$, the verifier decides that the message is authentic and equals its original value. MAC differs from digital signatures as MAC values are both generated and verified using the same secret key. This implies that the sender and receiver of a message must agree on the same key before initiating communications, as is the case with symmetric encryption. For the same reason, MAC does not provide the property of non-repudiation offered by signatures [4].

PRNG Generating random bit sequences is an important problem in cryptography. The security of many cryptographic systems depends on the generation of unpredictable bit sequences. Such sequences are used, for example, in stream ciphers, digital signature schemes, key materials of encryption schemes, in challenge-response identification systems, and in many other cryptographic protocols. Hash functions are often used as pseudo-random functions. That is, they provide a deterministic mechanism for generating random-seeming bit streams from some input source without disclosing any information about the input. A typical use is generating cipher keying material after a Diffie-Hellman exchange. IKE uses HMAC for this purpose, as does TLS.

Data Integrity As noted above, hash functions can be used to produce fingerprints of files or messages. Sometimes, instead of digitally signing these fingerprints, the values are stored separately from the data. This permits later detection of changes to the original data. One

system in which this is used is Tripwire. Tripwire is used as host intrusion detection system. Critical system files are fingerprinted; at intervals thereafter, the stored fingerprints are compared to values newly-calculated on the running system. If the message is tampered with, the digital fingerprint will change to reflect changes in the content. Therefore, the properties of cryptographic hash functions can be used to verify that files have not been altered; one can quickly determine data integrity.

3. RATIONALE OF RESEARCH

Several hash functions have been proposed, but most of them have been shown to be cryptographically weak. The most widespread hash functions are MD5 and SHA-1. Both the hash functions are built by iterating a compression function according to the Merkle-Damgård method [20]. Since their publications both hash functions have spread as cryptographic hash standards and have been deployed in a wide variety of security applications. Security of both hash functions have been damaged by recent dedicated attacks against internal structure of these hash functions and generic attacks against Merkle-Damgård construction. This necessitates the transition to newer more secure hash functions that replace today's weak hash functions.

Problem statement

To overcome structural and algorithmic weaknesses of existing MD4 based hash designs SHA-1 and NFORK such that the new hash designs can withstand the differential and generic attacks.

4. RESEARCH CONTRIBUTION

4.1 MDA-192 : A 192-bit design proposal

Design goal

The goal of this design is to provide mid-term security through some algorithmic modifications in the design of SHA-1.

Design description

MDA-192 is a dedicated hash function that takes a message less than 2^{64} bits in length and computes a 192-bit hash value. It uses the Merkle-Damgård iterative structure. The input

message is padded and divided into 512-bit blocks $M_1 \ldots M_t$. Each iteration takes a 192-bit chaining variable and a new message block, employs a compression function and produces the next chaining variable. The initial chaining value is a specified constant and the final chaining value is used as the output. The compression function processes one 512-bit message block per iteration. The 512-bit message is parsed into 16 32-bit words $W_0 \ldots W_{15}$, which are then expanded to 96 words using the following relation:

$$\begin{cases} (W_{i-3} \oplus W_{i-8} \oplus W_{i-14} \oplus W_{i-16}) \oplus ((W_{i-1} \oplus W_{i-2} \oplus W_{i-15}) <<< 13), \text{for } 16 \le i \le 51 \\ (W_{i-3} \oplus W_{i-8} \oplus W_{i-14} \oplus W_{i-16}) \oplus ((W_{i-1} \oplus W_{i-2} \oplus W_{i-15} \oplus W_{i-20}) <<< 13), \text{for } 52 \le i \le 95 \end{cases}$$

The compression function operates on a register of six 32-bit words $(A_i, B_i, C_i, D_i, E_i, F_i)$, initially loaded with the previous chaining value. For the first application of the compression function these words are initialized as:

$A_0 = 0\text{x}67452301$
$B_0 = 0\text{x}\text{EFCDAB89}$
$C_0 = 0\text{x}98\text{BADCFE}$
$D_0 = 0\text{x}10325476$
$E_0 = 0\text{x}\text{C3D2E1F0}$
$F_0 = 0\text{x}50\text{A28BE6}$

The state is modified over 4 rounds, each consisting of 24 steps of the following process:

$$F_{i+1} = E_i \oplus D_i$$
$$E_{i+1} = D_i \oplus C_i$$
$$D_{i+1} = C_i \oplus B_i$$
$$C_{i+1} = B_i <<< 30$$
$$B_{i+1} = A_i \oplus (W_i <<< 15)$$
$$A_{i+1} = ((A_i <<< 5) + f_i(B_i, C_i, D_i) + F_i + W_i + k_i) <<< (W_i \bmod 32)$$

The Boolean functions f_i and the constants k_i are specified in Table 1. Upon the completion of the compression function the output is the concatenation of bits of $(A_0 + A_{96}), (B_0 + B_{96}), (C_0 + C_{96}), (D_0 + D_{96}), (E_0 + E_{96}), (F_0 + F_{96})$.

Security analysis

The difficulty of producing any message X having a given hash value $h(X)$ is of the order of 2^{192} operations. The difficulty of finding any message Y, $Y \neq X$ with $h(Y) = h(X)$, when

X and $h(X)$ are given, is of the order of 2^{192} operations. The difficulty of producing two distinct messages X and Y having the same hash value $h(X) = h(Y)$ is of the order of 2^{96} operations. The message expansion of MDA-192 is extended to 96^{th} step. Minimum Hamming distance between two set of message words is at least 112 which is far better than SHA-1. It provides higher minimum distance between similar words, high randomness, good mixing of bits and lesser control over the propagation of difference in the words. By using message words heavily in variable bit rotations and computation of step operations, we introduce the redundancy in the round function of MDA-192. Thus hash results are more strongly dependent on the input message. Also working variables are XORed to update values of chaining variables in step operations to provide good diffusion. There are 24 steps in each round of MDA-192. Extra 4 steps in each round provide high randomness and good mixing of message bits. It has equal distribution of 0s and 1s. This guarantees equal distribution of 0s and 1s in final output of compression function. Differential collision characteristics of modified message expansion has been estimated and found that the minimum distance in the last eighty words is estimated to at least 112. Thus, we expect our proposed compression function to have a differential collision characteristic of probability close to $2^{-2.5\times112}$.

Table 1: Constants and Boolean functions used in MDA-192

Round	Step i	$f_i(B,C,D)$	k_i
1	0-23	$f_{IF} = (B \wedge C) \vee (\bar{B} \wedge D)$	0x5A827999
2	24-47	$f_{XOR} = (B \oplus C \oplus D)$	0x6ED9EBA1
3	48-71	$f_{MAJ} = (B \wedge C) \vee (B \wedge D) \vee (C \wedge D)$	0x8F1BBCDC
4	72-95	$f_{XOR} = (B \oplus C \oplus D)$	0xCA62C1D6

4.2 DSHA-1 : A 160-bit design proposal

Design goal

The goal of this design is to provide stronger algorithm to overcome the structural flaws of SHA-1.

Design description

DSHA-1 is a 160-bit dedicated hash function based on the design principle of SHA-1. It takes as input a message with maximum length 2^{64}-1 bits and returns a 160-bit hash value. It applies the dither construction to a dedicated compression function. The input message is padded and split into t 512-bit message blocks. It is padded using following method. First a 1

is appended and then padded with 0's until the length of the new message is congruent to 448 modulo 512; finally the length of the message is inserted in the last 64-bits of the message. At each iteration of the compression function *f*, a 160-bit chaining variable H_k is updated using a message block and a dither input, i.e. $H_{k+1} = f(H_k, M_{k+1}, R_{k+1})$. The initial value H_0 is predefined and H_t is the output of the hash function. The compression function of DSHA-1 is composed of 80 steps; each processing a 32-bit message word W_i to update a 160-bit buffer consists of 5 32-bit internal registers (A, B, C, D, E). These five registers are initialized with the following hexadecimal values:

$A_0 = 0\text{x}67452301$
$B_0 = 0\text{x}\text{EFCDAB89}$
$C_0 = 0\text{x}98\text{BADCFE}$
$D_0 = 0\text{x}10325476$
$E_0 = 0\text{x}\text{C3D2E1F0}$

Since more message bits than available are utilized, a message expansion is therefore defined. In message expansion the 512-bit input message M_k, consisting of 16 32-bit words $W_0 \ldots W_{15}$ is linearly expanded to 80 32-bit words as follows:

$$\begin{cases} (W_{i-3} \oplus W_{i-8} \oplus W_{i-14} \oplus W_{i-16}) \oplus ((W_{i-1} \oplus W_{i-2} \oplus W_{i-15}) <<< 13), \text{for } 16 \le i \le 35 \\ (W_{i-3} \oplus W_{i-8} \oplus W_{i-14} \oplus W_{i-16}) \oplus ((W_{i-1} \oplus W_{i-2} \oplus W_{i-15} \oplus W_{i-20}) <<< 13), \text{for } 36 \le i \le 79 \end{cases}$$

Dither inputs are generated through a pseudorandom number generator. Total 80 32-bit pseudo-random numbers are generated. R_k consists of 80 32-bit pseudo-random numbers $r_0 \ldots r_{79}$. These random numbers are given to compression function *f* as a third input. Each step operation makes the use of exact one 32-bit pseudo-random number as dither input.

The compression function is composed of four rounds of processing where each round is made up of twenty steps. The following processing of a compression function is applied 80 times:

$E_{i+1} = D_i$
$D_{i+1} = C_i$
$C_{i+1} = B_i <<< 30$
$B_{i+1} = A_i$
$A_{i+1} = (A_i <<< 5) + F_i(B_i, C_i, D_i) + E_i + W_i + K_i + r_i$

Upon the completion of the compression function the output is obtained as: $(A_0 + A_{80}), (B_0 + B_{80}), (C_0 + C_{80}), (D_0 + D_{80}), (E_0 + E_{80})$. Where K_i are predetermined constants and F_i are Boolean functions defined in Table 2.

Security analysis

The difficulty of producing any message X having a given hash value $h(X)$ is of the order of 2^{160} operations. The difficulty of finding any message Y, $Y \neq X$ with $h(Y) = h(X)$, when X and $h(X)$ are given, is of the order of 2^{160} operations. The difficulty of producing two distinct messages X and Y having the same hash value $h(X) = h(Y)$ is of the order of 2^{80} operations. The design of the proposed hash function is secure against generic attacks. The fixed point attack is applied to hash functions that use a compression function. A fixed point for the hash function H is a pair (h, m), such that f (h, m) = h. This means that the message block m does not affect the result of the hash value and chaining value h remains same after iteration with message block m. Thus, message block m can be used to obtain second pre-image attacks. More generalized form of this attack is a length extension attack. In this attack the adversary searches for pairs of collision of different length. It is based on the principle that If hash values of messages M and M' collide then appending a common suffix also leads to a collision. An extra input, dither to the compression function makes the chaining variables repetition free, each time there will be a new outcome from a compression function so it is difficult to find fixed point attack and length extension attack against compression function of DSHA-1. Joux provided a method to find multi-collisions in iterated hash function. The generic multi-collision attack against the compression function of DSHA-1 would require 2^{80} operations. Thus, the construction of proposed hash function does not suffer from major generic attacks. We have also analyzed the security of the proposed design against near-collision attack using chosen prefix method and found that it has strong resistance against it.

Table 2: Constants and Boolean functions used in DSHA-1

Round	Step i	$F_i(B,C,D)$	K_i
1	0-19	$F_{IF} = (B \wedge C) \vee (\bar{B} \wedge D)$	0x5A827999
2	20-39	$F_{XOR} = (B \oplus C \oplus D)$	0x6ED9EBA1
3	40-59	$F_{MAJ} = (B \wedge C) \vee (B \wedge D) \vee (C \wedge D)$	0x8F1BBCDC
4	60-79	$F_{XOR} = (B \oplus C \oplus D)$	0xCA62C1D6

4.3 MNF-256: A 256-bit design proposal

Design goal

The aim for designing the MNF-256 hash algorithm is to make NFORK more efficient and strong against generic attacks.

Design description

MNF-256 is an iterative hash function based on the dither construction that processes 512-bit input message blocks and produces a 256-bit hash value. The compression function consists of 3 parallel branches which we denote $BRANCH_j; j = 1,2,3$. In each branch the state variables are updated according to the different message word permutation and combined with the chaining variables after the last step. Each one of three branches using a different permutation σ_j of 16 message words $M_i \quad i = 0,\ldots,15$ and the same set of chaining variables $CV = (A,B,C,D,E,F,G,H)$.the compression function updates the set of chaining variables according to the following relation:

$$CV_{i+1} = CV_i + \{[BRANCH_1(CV_i, M, D) + BRANCH_2(CV_i, M, D)] \oplus [BRANCH_2(CV_i, M, D) + BRANCH_3(CV_i, M, D)]\},$$

where modular and XOR additions are performed word-wise. Each branch function $BRANCH_j, j = 1,2,3$ consists of eight steps. In each step $k = 0,\ldots,7$ branch function updates its own copy of eight chaining variables according to the following equations:

$$A_{j,k+1} = H_{j,k} \oplus (p_4 <<< 8) \oplus d_{j,k},$$
$$B_{j,k+1} = A_{j,k} + M_{\sigma_j(2k)} + \alpha_{j,k} \oplus d_{j,k},$$
$$C_{j,k+1} = B_{j,k} + p_1 \oplus d_{j,k},$$
$$D_{j,k+1} = C_{j,k} + (p_1 <<< 13) \oplus p_2 \oplus d_{j,k},$$
$$E_{j,k+1} = D_{j,k} \oplus (p_2 <<< 17) \oplus d_{j,k},$$
$$F_{j,k+1} = E_{j,k} + M_{\sigma_j(2k+1)} + \beta_{j,k} \oplus d_{j,k},$$
$$G_{j,k+1} = F_{j,k} + p_3 \oplus d_{j,k},$$
$$H_{j,k+1} = G_{j,k} + (p_3 <<< 3) \oplus p_4 \oplus d_{j,k}.$$

Where,

$$p_1 = f(A_{j,k} + M_{\sigma_j(2k)}), p_2 = g(A_{j,k} + M_{\sigma_j(2k)} + \alpha_{j,k})$$
$$p_3 = g(E_{j,k} + M_{\sigma_j(2k+1)}), p_4 = f(E_{j,k} + M_{\sigma_j(2k+1)} + \beta_{j,k})$$

$$A_{j,k+1} = H_{j,k} \oplus (p_4 <<< 8) \oplus d_{j,k},$$
$$B_{j,k+1} = A_{j,k} + M_{\sigma_j(2k)} + \alpha_{j,k} \oplus d_{j,k},$$
$$C_{j,k+1} = B_{j,k} + p_1 \oplus d_{j,k},$$
$$D_{j,k+1} = C_{j,k} + (p_1 <<< 13) \oplus p_2 \oplus d_{j,k},$$
$$E_{j,k+1} = D_{j,k} \oplus (p_2 <<< 17) \oplus d_{j,k},$$
$$F_{j,k+1} = E_{j,k} + M_{\sigma_j(2k+1)} + \beta_{j,k} \oplus d_{j,k},$$
$$G_{j,k+1} = F_{j,k} + p_3 \oplus d_{j,k},$$
$$H_{j,k+1} = G_{j,k} + (p_3 <<< 3) \oplus p_4 \oplus d_{j,k}.$$

Where $R_{j,i}$ denotes the value of the register $R \in A, B, \ldots, H$ in the j^{th} branch after step *i* and for all branches chaining variables are initialized as:

$$A_0 = 0x6A09E667 \quad B_0 = 0xBB67AE85$$
$$C_0 = 0x3C6EF372 \quad D_0 = 0xA54FF53A$$
$$E_0 = 0x510E527F \quad F_0 = 0x9B05688C$$
$$G_0 = 0x1F83D9AB \quad H_0 = 0x5BE0CD19$$

Function *f* and *g* defined as

$$f(x) = x \oplus (x <<< 15 \oplus x <<< 27)$$
$$g(x) = x \oplus (x <<< 7 + x <<< 25).$$

Constants $\delta_0, \ldots, \delta_{15}$ are defined as the first 32 bits of fractional parts of binary expansions of cube roots of the first 16 primes and are presented in Table 3.

Security analysis

The difficulty of producing any message X having a given hash value $h(X)$ is of the order of 2^{256} operations. The difficulty of finding any message Y, $Y \neq X$ with $h(Y) = h(X)$, when X and $h(X)$ are given, is of the order of 2^{256} operations. The difficulty of producing two distinct messages X and Y having the same hash value $h(X) = h(Y)$ is of the order of 2^{128} operations. To construct a differential characteristic with a high probability for a branch function, say $BRANCH_i$ and then expects that, the operation of the output differences in the other branches Δ_3 is equal to Δ_1. Proposed hash function is secure against this strategy because the outputs of each branch function are random; the probability of the event is almost close to 2^{256}. To insert the message difference which yields same message difference pattern in all the three branches and expect that, same differential characteristics occur simultaneously in three branches. However, using the message word reordering this can be avoided. Moreover, using different operators highly complicates the computation of good

differential paths. Addition of message words, parallel mixing structure, rotation of registers and addition of dither value made compression function stronger against different attacks. The design is further improved by constructing it on HAIFA. We have also analyzed the security of the proposed design against meet-in-the-middle technique and found that it is secure against it.

Table 3: Constants used in MNF-256

$\delta_0 = 0x428A2F98$	$\delta_4 = 0x3956C25B$	$\delta_8 = 0xD807AA98$	$\delta_{12} = 0x72BE5D74$
$\delta_1 = 0x71374491$	$\delta_5 = 0x59F111F1$	$\delta_9 = 0x12835B01$	$\delta_{13} = 0x80DEB1FE$
$\delta_2 = 0xB5C0FBCF$	$\delta_6 = 0x923F82A4$	$\delta_{10} = 0x243185BE$	$\delta_{14} = 0x9BDC06A7$
$\delta_3 = 0xE9B5DBA5$	$\delta_7 = 0xAB1C5ED5$	$\delta_{11} = 0x550C7DC3$	$\delta_{15} = 0xC19BF174$

Table 4: Constants permutation used in MNF-256

K	$\alpha_{1,k}$	$\beta_{1,k}$	$\alpha_{2,k}$	$\beta_{2,k}$	$\alpha_{3,k}$	$\beta_{3,k}$
0	δ_0	δ_1	δ_{15}	δ_{14}	δ_1	δ_0
1	δ_2	δ_3	δ_{13}	δ_{12}	δ_3	δ_2
2	δ_4	δ_5	δ_{11}	δ_{10}	δ_5	δ_4
3	δ_6	δ_7	δ_9	δ_8	δ_7	δ_6
4	δ_8	δ_9	δ_7	δ_6	δ_9	δ_8
5	δ_{10}	δ_{11}	δ_5	δ_4	δ_{11}	δ_{10}
6	δ_{12}	δ_{13}	δ_3	δ_2	δ_{13}	δ_{12}
7	δ_{14}	δ_{15}	δ_1	δ_0	δ_{15}	δ_{14}

Table 5: Message permutation used in MNF-256

t	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\sigma_1(t)$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\sigma_2(t)$	14	15	11	9	8	10	3	4	2	13	0	5	6	7	12	1
$\sigma_3(t)$	7	6	10	14	13	2	9	12	11	4	15	8	5	0	1	3

Table 6: Dither input permutation used in MNF-256

k	0	1	2	3	4	5	6	7
$d_{1,k}$	d_0	d_2	d_4	d_6	d_8	d_{10}	d_{12}	d_{14}
$d_{2,k}$	d_{15}	d_{13}	d_{11}	d_9	d_7	d_5	d_3	d_1
$d_{3,k}$	d_{14}	d_{12}	d_{10}	d_8	d_6	d_4	d_2	d_0

Table 7: Comparison of proposed designs with existing designs

Property	SHA-1	DSHA-1	MDA-192	NFORK	MNF-256
Input parameters	2	3	2	2	3
Output parameter	1	1	1	1	1
Construction	MerkleDamgård	Dither	MerkleDamgård	MerkleDamgård	Dither
Message block size	512-bit	512-bit	512-bit	512-bit	512-bit
Word size	32-bit	32-bit	32-bit	32-bit	32-bit
Total input bits to the compression function	2720-bit	5280-bit	3264-bit	768-bit	1280-bit
Output hash value	160-bit	160-bit	192-bit	256-bit	256-bit
Number of branches	---	---	--	4	3
Message blocks in step	1	1	1	2	2
Constants	4	4	4	16	16
Number of steps	80	80	96	8	8
Shift values	fixed	fixed	variable	fixed	fixed
Non-linear functions	4	4	4	2	2
Bijective function	--	--	--	present	present
Used operations	$+, \lll, \oplus$	$+, \lll, \oplus$	$+, \lll, \oplus$	$+, \lll, \oplus$	$+, \lll, \oplus$
Efforts required to find preimage	2^{160}operations	2^{160}operations	2^{192}operations	2^{256} operations	2^{256} operations
Efforts required to find 2^{nd}-preimage	2^{160}operations	2^{160}operations	2^{192} operations	2^{256} operations	2^{256} operations
Efforts required to find collision	$<2^{80}$operations	2^{80}operations	2^{96} operations	$<2^{128}$ operations	2^{128} operations
Differential cryptanalysis	simple	complex	complex	complex	complex
Generic attacks	possible	not possible	possible	possible	not possible

5 TESTING METHODS AND RESULT ANALYSIS

Hash functions are used in a number of cryptographic applications and protocols such as digital signatures, message authentication, data integrity, password protection, SSL, and pseudo-random number generation. The strength of these applications and protocols relies on the quality of the cryptographic hash functions used. Basic properties of hash functions required include easiness in computing the digital fingerprints and hardness in computing the data from a given digital fingerprint and in finding another data with the same hash value. However, these characteristics are not sufficient for the cryptographic hash functions to be applied in cryptographic applications. One of the most basic properties expected from cryptographic hash functions is passing statistical randomness testing. In addition to the randomness of hash values, hash functions should satisfy collision resistance property as well. According to collision resistance property, it should be infeasible to find any pair of messages which result to the same hash value. The quality of the hash function is highly influenced by random output and collision resistance property.

The avalanche criterion is used to assess the ability of hash function to distribute hash values randomly. The avalanche criterion measures the influence of hash input to hash value bits. Avalanche is the property that with the change of one input bit all output bits change with a probability of 50%. The closer this avalanche criteria is fulfilled the more random are the hash values distributed. Corresponding to above mentioned criteria we conducted following tests:

5.1 Test for 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 $1 \le i \le 512$. Then we generated hash values of all these new messages and finally computed Hamming distances or changed bit numbers between hash values of original message and modified messages. Ideally it should be half of the length of hash value. Corresponding frequency distributions of distances for different algorithms have been shown in Figure 1 to Figure 5 and different ranges of distances have been listed in Table 8 to Table 10. Most of the distances lie in the range of 80-100 for DSHA-1, 96-125 for MDA-192 and 128-160 for MNF-256. Results indicate that proposed algorithms have good randomness.

Table 8 Range of distances for SHA-1 and DSHA-1

	SHA-1		DSHA-1	
Distances	Hash pairs	Percentage (%)	Hash pairs	Percentage (%)
80±5	306	59.76	328	64.06
80±10	467	91.29	471	91.99
80±15	499	97.46	507	99.02

Table 9: Range of distances for MDA-192

Distances	Hash pairs	Percentage (%)
96±5	260	50.78
96±10	412	80.46
96±15	484	94.53

Table 10: Range of distances for NFORK and MNF-256

	NFORK		MNF-256	
Distances	Hash pairs	Percentage (%)	Hash pairs	Percentage (%)
128±5	231	45.11	263	51.36
128±10	383	74.81	428	83.59
128±15	476	92.96	484	94.53

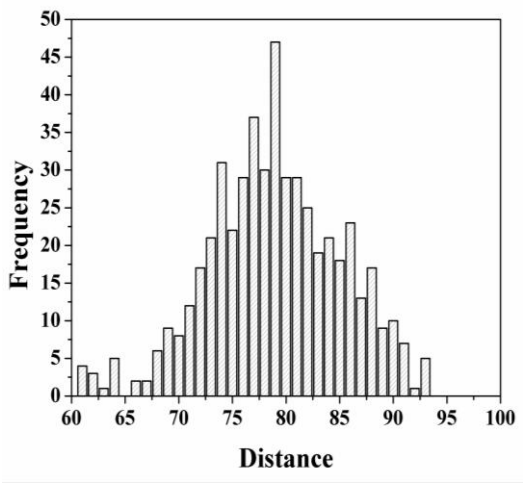

Figure 1: Frequency distribution for SHA-1

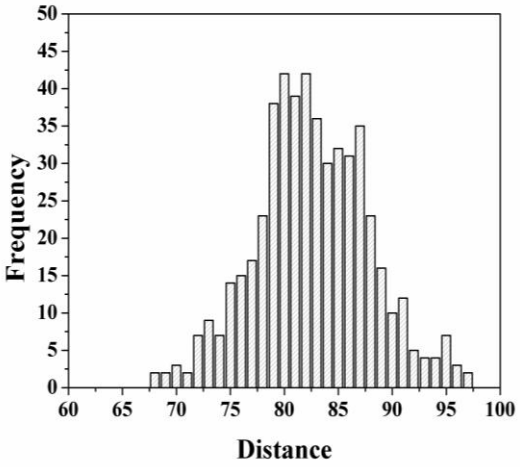

Figure 2: Frequency distribution for DSHA-1

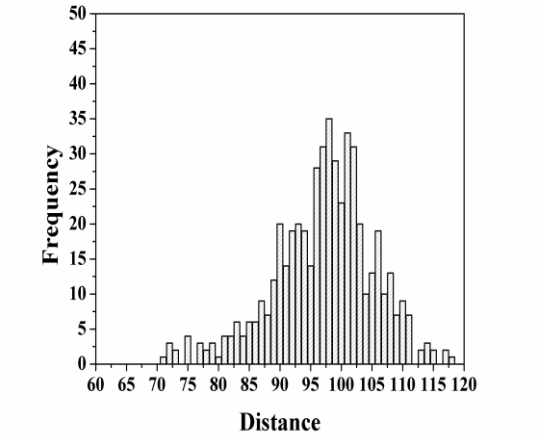

Figure 3: Frequency distribution for MDA-192

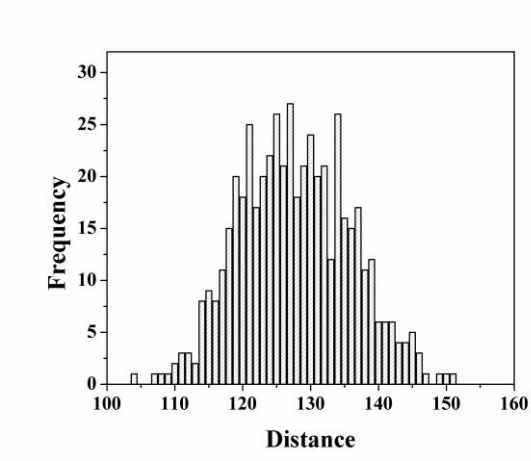

Figure 4: Frequency distribution for NFORK

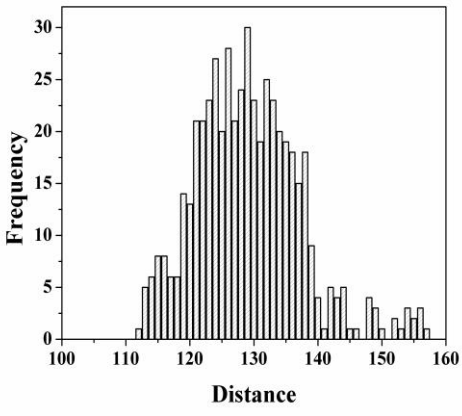

Figure 5: Frequency distribution for MNF-256

5.2 The bit variance test

It 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 $1 \le i \le n$, 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 one input message bit change. Results for different algorithms for bit variance test have been shown in Table 11. Results show that all the proposed hash functions have attained maximum performance and achieved avalanche criterion.

5.3 Statistical diffusion test

Diffusion is the primary engineering design principle for cryptographic hash functions. Diffusion means spreading out of the influence of a single plaintext bit so as to hide the statistical structure of the plaintext. However, for hash functions it is the statistical irrelevance between the input bits and the hash value. Strong diffusion capability can be achieved by making each bit of the input affect each bit of the hash value. More formally, a hash function is said to have strong diffusion capability, if given M, M' and $h(M) = H$, $h(M') = H'$ it is highly impractical to reveal the relation between H, H' where M and M' may differ by even only a single bit. Thereby, any single bit change in the input would cause a drastic change in the hash value.

We have performed the following diffusion test. 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. Two hash values are compared with each other and the number of changed bit is counted as B_i. This kind of test is performed N = 2048 times. We used four statistics for this: mean changed bit number $\bar{B}$, mean changed probability P, standard deviation of the changed bit number ΔB and standard deviation ΔP.

Mean changed bit number: $\bar{B} = \frac{1}{N}\sum_{i=1}^{N} B_i$

Mean changed probability: $P = (\bar{B}/n) \times 100\%$

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

where, N is total statistic number. n represents the length of hash value. The corresponding distribution of changed bit number for first 512 samples is shown as Figure 6 to Figure 10. ΔB and ΔP indicate the stability of diffusion. From the results in Table 12, for proposed designs ΔB and ΔP are very small and mean changed bit number and probability are well above the ideal values {80, 96, 128} and 50%. This indicates stable diffusion capability for proposals.

5.4 Collision test

Collision resistance is an important design criterion for hash functions, which means that it should be hard to find two messages with the same hash value. In order to investigate the collision resistance capability of the hashing approach, 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 then generated and stored in the same format. Two hash values are compared with each other and the number of character in this format with the same value at the same location in hash value is counted. The absolute difference of the two hash result 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. Results are shown in Table 13. All the proposed designs have achieved the ideal value 85.33.

In the second experiment, the hash value for a randomly chosen message is generated and stored in ASCII format similarly. This experiment concentrates on the possibility of colliding between every two hash results, thus every two hash results should be compared. The simulation is performed 2048 times. The maximum number of equal entries for all proposals is 2. Besides, most of the entries are different in ASCII format. It shows that the all the proposals possess a strong collision resistance capability.

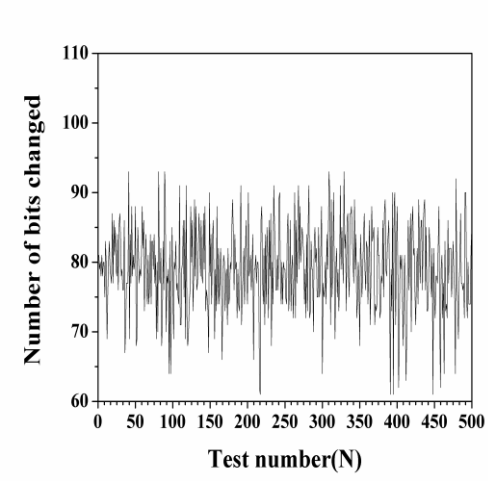

Figure 6: Distribution of changed bit number for SHA-1

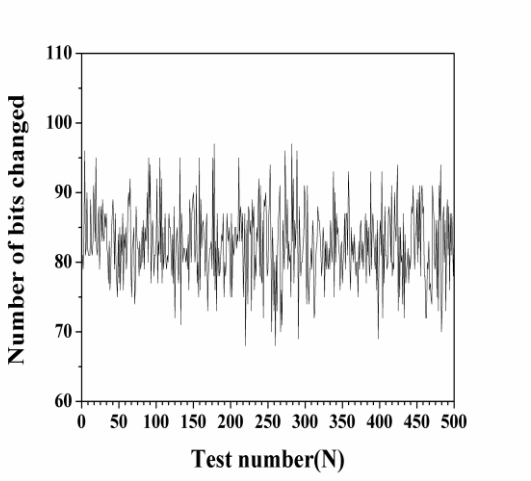

Figure 7: Distribution of changed bit number for DSHA-1

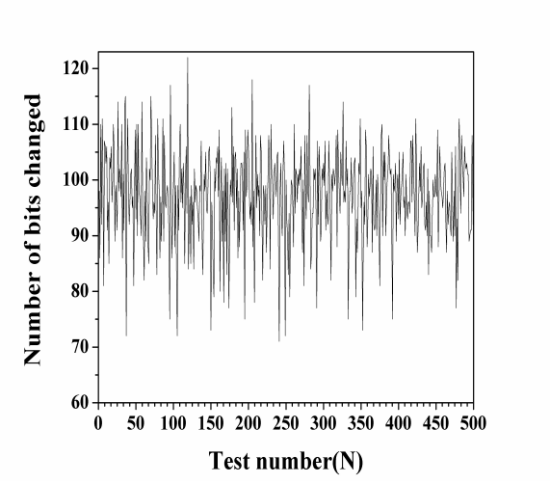

Figure 8: Distribution of changed bit number for MDA-192

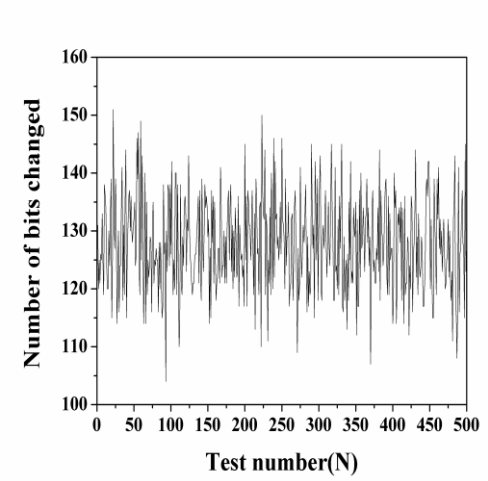

Figure 9: Distribution of changed bit number for NFORK

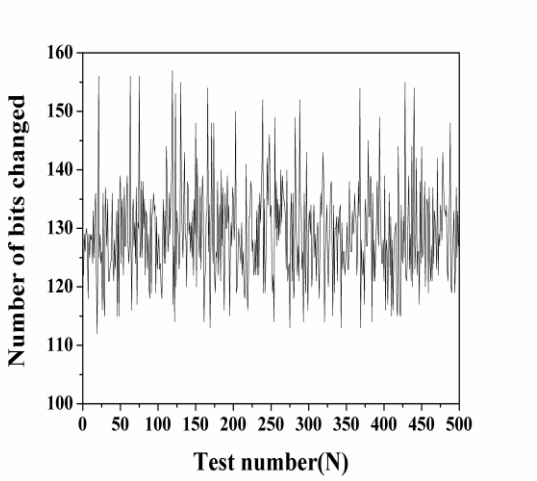

Figure 10: Distribution of changed bit number for MNF-256

Table 11: Results for Bit variance test

Hash function	Mean frequency of 1s (expected)	Mean frequency of 1s (calculated)
SHA-1	256.50	249.11
DSHA-1	256.50	256.67
MDA-192	256.50	257.03
NFORK	256.50	251.12
MNF-256	256.50	257.15

Table 12: Statistics of number of changed bits

Algorithm	$\bar{B}$	$P(\%)$	ΔB	$\Delta P(\%)$
SHA-1	79.3926	49.6203	6.3472	3.6452
DSHA-1	80.4171	50.2606	6.1707	3.8567
MDA-192	96.1928	50.1004	8.6174	4.4882
NFORK	127.8752	49.9512	10.4495	4.0818
MNF-256	129.1615	50.4537	8.9771	3.5066

Table 13: Absolute differences of two hash values

Algorithm	Max	Min	Mean	Mean/char
SHA-1	2332	695	1642.63	82.13
DSHA-1	2834	913	1724.40	86.22
MDA-192	3155	927	2068.11	86.17
NFORK	3178	1686	2658.31	83.07
MNF-256	3650	1833	2752.74	86.02

5.5 Speed test

The speed test has been carried out over an Intel Pentium 4 CPU at 1.47 GHz according to the following procedure- We select a message size *S* (in bytes) and generate 1000 random messages of size *S*. The hash function is applied to each of these 1000 messages, measuring the time required to compute each of them. Finally, we take the average over 1000 samples. This process is applied to all five algorithms. The average CPU computation times (in sec) obtained for SHA-1, DSHA-1, MDA-192, NFORK and MNF-256 are listed in Table 14. It was found that proposed DSHA-1 and MDA-192 takes extra time for the computation of hash value than SHA-1. Since single block compression function of DSHA-1 takes extra dither input of 2560-bit for all 80 steps, it results in excess time to compute the final hash value. This could be optimized by inserting dither input bits (either 1-bit or 2-bit) to the different

message sub blocks itself and values of these input bits may depends upon index value of particular message sub block but this could lead to diluting the desired level of dithering. Simulation results in Table 14 also imply that MNF-256 is faster than NFORK.

Table 14: Computation times

S (in bytes)	SHA-1 (in sec)	DSHA-1 (in sec)	MDA-192 (in sec)	NFORK (in sec)	MNF-256 (in sec)
100	0.0032	0.0518	0.0739	0.0238	0.0221
1000	0.0298	0.1658	0.1685	0.0716	0.0685
10000	0.5821	1.4283	1.7934	0.3519	0.2961
100000	5.6352	15.0813	16.6894	0.9421	0.7193

5.6 Crypto-Precision: Testing tool

Crypto-Precision is designed to be adaptable to various cryptographic hash functions and different hash function construction methods. We have tested three hash functions; SHA-1, FORK-256 and NFORK. These algorithms have can be implemented on five different constructions: Merkle-Damgård with Permutation (MDP), Dither, HAIFA, Dither with MDP and HAIFA with MDP. The current version of the tool allows its user to dynamically generate code for hash functions based on specific algorithm and construction. The generated hash code undergoes a generic robustness testing module with a set of representative test cases, where their randomness, collision resistance and speed are observed for further analysis.

6. CONCLUSION

In the first proposal a hash function MDA-192 presented which process a message of arbitrary length by 512-bit blocks and produces as output a 192-bit hash value or message digest. Message expansion of the algorithm expands 16, 32-bit words to 96, 32-bit words. Step operation of the proposed algorithm is more dependent on the message words. The message expansion step operations provide good mixing of bits which make algorithm stronger against the differential cryptanalysis.

In the second proposal, a hash function is presented that follows design principle of SHA-1 and 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. No brute-force preimage and second-preimage attack against proposal is known for much less than 2^{160} operations.

In the third proposal, a hash function MNF-256 with larger digest size (256-bit) is given based on the design principle of NFORK. It takes 512-bit message blocks and generates 256 -bit hash value. A random sequence is added as an additional input to the compression function of MNF-256. A new hash function has been designed with improved security and reasonable speed. Its core strength is due to the factors like three parallel lines with 256-bit hash length, different message ordering in each branch, and extra input to compression function. These enhancements make MNF-256 capable of resisting attacks to a much higher degree compared hash functions with similar design. It is proven that MNF-256 is secure against any known attacks on hash functions.

7. THESIS OUTLINE

Thesis is organized into total eight chapters. **Chapter 1** introduces the role of cryptography & cryptographic hash functions in the secure information communication and establishes the need for inventing new hash algorithms. **Chapter 2** first, critically reviews MD construction method along with its various variants. There after a comparative view of common compression methods, and generic attack methods are discussed in short. We also surveyed the design evolution of dedicated hash function families and various attacks found on them. **Chapter 3** presents our first serial structured hash design MDA-192 which is based on SHA-1 and built on Merkle-Damgård construction method. The design focus is how to improve the message expansion and algorithmic steps of SHA-1 such that better security level against differential attack and brute force attack can be achieved. **Chapter 4** focuses on our second proposal DSHA-1 which is also based on SHA-1 but built on dither construction method. **Chapter 5** proposes a parallel structured MNF-256 hash function which is based on NFORK and built on dither construction method. The design goals of this proposal is how to optimize so far known the best secured parallel structured hash functions such that resultant design can shows better computation speed as well more robustness against generic attacks, in comparison to SHA-1 and NFORK. **Chapter 6** presents the security analysis of the MDA-192, DSHA-1 and MNF-256 against various generic attacks. **Chapter 7** comments on

comparative performance evaluation of proposed hash functions with their parent hash functions according to laid-down various statistical benchmarks. **Chapter 8** presents concluding remarks with possible future research directions.

REFERENCES

[1] Bakhtiari S., Safavi-Naini R., and Pieprzyk J., “Cryptographic Hash Functions: A Survey”, Technical Report 95-09, Department of Computer Science, University of Wollongong, 1995.

[2] Menezes A. J., van Oorschot P. C., and Vanstone S. A., Handbook of Applied Cryptography. CRC Press, 1997.

[3] Schneier B., Applied Cryptography. New York: John Wiley & Sons, 1996.

[4] Bellare M., Canetti R. and Krawczyk H., "Keying Hash Functions for Message Authentication", CRYPTO ’96, pp. 1-15, 1996.

[5] Rivest R., “The MD4 Message Digest Algorithm”, Request for Comments (RFC) 1320, Internet Engineering Task Force, 1992. http://www.rfceditor.org/rfc/pdfrfc/rfc1320.txt.pdf.

[6] Rivest R., “The MD5 Message Digest Algorithm”, Request for Comments (RFC) 1321, Internet Engineering Task Force, 1992.

[7] NIST, Secure Hash Standard (SHS). Federal Information Processing Standards 180. 1993.

[8] NIST, Secure Hash Standard (SHS), Federal Information Processing Standards 180-1, 1995.

[9] NIST, Secure Hash Standard (SHS). Federal Information Processing Standards 180-2. 2002.

[10] Preneel B, Bosselaers A, Dobbertin H., “RIPEMD-160: A Strengthened Version of RIPEMD”, FSE’96, LNCS, vol. 1039, pp. 71–82, 1997.

[11] Dobbertin H., “Cryptanalysis of MD4”, Journal of Cryptology, vol. 11, pp. 253-271, 1998.

[12] Wang X, Feng D, Lai X, Yu H., “Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD”, Cryptology ePrint Archive, Report 2004/199, 2004.

[13] Wang X, Yin Y L, and Yu H. , “Finding Collisions in the Full SHA-1”, CRYPTO’05, LNCS, vol. 3621, pp. 17-36, 2005.

[14] Matusiewicz K, Contini S, Pieprzyk J., “Weaknesses of the FORK-256 Compression Function”, Cryptology ePrint Archive, Report 2006/317,2006.

[15] Mendel F, Lano J, Preneel B., “Cryptanalysis of Reduced Variants of the FORK-256 Hash Function”, CT-RSA 2007, LNCS, vol. 4377, pp. 85–100, 2006.

[16] Danda M., Design and Analysis of Hash Functions, Thesis, Victoria University, 2007.

[17] O. Saarinen Markku-Juhani, "A Meet-in-the-Middle Collision Attack against the New FORK-256", INDOCRYPT'07, LNCS, vol. 4859, pp. 10-17, 2007.

[18] Preneel B., "The First 30 Years of Cryptographic Hash Functions and the NIST SHA-3 Competition", CT-RSA'10, LNCS, vol. 5985, pp. 1-14, 2010.

[19] Preneel B.**, "**The NIST SHA-3 Competition: A Perspective on the Final Year", AFRICACRYPT'11, LNCS, vol. 6737, pp. 383-386, 2011.

[20] Damgård I., "A Design Principle for Hash Functions", CRYPTO'89, LNCS, vol. 435, pp. 416-427, 1989.

[21] Rivest R., "Abelian Square-free Dithering for Iterated Hash Functions", ECRYPT Hash Function Workshop, 2005, online available at: http://csrc.nist.gov/groups/ST/hash/documents/rivest-asf-paper.pdf.

[22] Hong, D., Chang, D., Sung, J., Lee, S., Hong, S., Lee, J., Moon, D., and Chee, S., "NewFORK-256", Cryptology ePrint Archive, Report 2007/185, 2007.

[23] Keccak Design Team, The Keccak Sponge Function Family, http://keccak.noekeon.org/.

[24] Joux A., "Multicollisions in Iterated Hash Functions: Application to Cascaded Constructions", CRYPTO '04, LNCS, vol. 31252, pp. 306-316, 2004.

[25] Dean R. D., Formal Aspects of Mobile Code Security, PhD thesis, Princeton University, 1999.

[26] Kelsey J., Kohno T., "Herding Hash Functions and the Nostradamus Attack", EUROCRYPT '06, LNCS, vol. 4004, pp. 183-200, 2006.

[27] Merkle R., "One Way Hash Functions and DES", CRYPTO '89, LNCS, vol. 435, pp. 428-446, 1989.

[28] Li Y., Deng S., Xiao D., "A Novel Hash Algorithm Construction Based on Chaotic Neural Network", Neural Computing and Applications, vol. 20, no. 1, pp. 133-141, 2011.

[29] Li Y., Xiao D., Deng S., Zhou G., " Improvement and Performance Analysis of a Novel Hash Function based on Chaotic Neural Network", Neural Computing and Applications, vol. 22, no. 2, pp. 391-402, 2013.

[30] Mihaljevie M., Zheng Y., Imai H., "A Cellular Automaton Based Fast One-Way Hash Function Suitable for Hardware Implementation", PKC '98, LNCS, vol. 1431, pp. 217-233, 1998.

[31] Biham E., Dunkelman O., "A Framework for Iterative Hash Functions–HAIFA", Cryptology ePrint Archive, Report2007/278, 2006.

[32] Hirose S., Park J. H., Yun A., "A Simple Variant of the Merkle-Damgård Scheme with a Permutation", CRYPTOLOGY - ASIACRYPT '07, LNCS, vol. 4833, pp. 113-129, 2007.

LIST OF PUBLICATIONS

[1] Tiwari H., Asawa K., "*A Secure Hash Function MD-192 with Modified Message Expansion*", International Journal of Computer Science and Information Security, vol. 7, no. 2, pp. 108-111, Feb. 2010. (Indexed in DBLP)

[2] Tiwari H., Asawa K., "*Cryptographic Hash Function: An Elevated View*", European Journal of Scientific Research, SRP, vol. 43, no. 4, pp. 452-465, Jun. 2010. (Indexed in Scopus)

[3] Tiwari H. et al., "*Crypto-Precision: Testing Tool for Hash Function*", SNDS'12, Communications in Computer and Information Science proceedings, Trivandrum, India, Springer, vol. 335, pp. 205-214, Oct. 2012. (Indexed in Scopus and DBLP)

[4] Tiwari H., Asawa K., "*A Secure and Efficient Cryptographic Hash Function Based on NewFORK-256*", Egyptian Informatics Journal, Elsevier, vol.13, no. 3, pp. 199-208, Nov. 2012. (Indexed in Scopus)

[5] Tiwari H., Asawa K., "*Enhancing the Security Level of SHA-1 by Replacing the MD Paradigm*", Journal of Computing and Information Technology, SRCE, vol. 21, no. 4, pp. 223-233, Nov. 2013. (Indexed in Scopus and DBLP)

[6] Tiwari H., Asawa K., "*Building a 256-bit Hash Function on a Stronger MD variant*", Central European Journal of Computer Science, Springer. (In press, Indexed in DBLP)