Design and Analysis of a New Hash Algorithm with Key Integration

Richa Purohit (Arya)
Amity University Rajasthan, Jaipur, India

Upendra Mishra, Ph. D
Amity University Rajasthan, Jaipur, India

Abhay Bansal, Ph. D
Amity University Uttar Pradesh Noida, India

ABSTRACT
Message Integrity and authenticity are the primary aim with the ever increasing network protocols' speed. Cryptographic Hash Functions are main building block of message integrity. Many types of hash functions are being used and developed. In this paper, we propose and describe a new keyed hash function. This newly designed function produces a hash code of 128 bits for an arbitrary length input. The function also uses a key during hashing, so any intruder that does not know key, cannot forge the hash code, and, thus it fulfills the purpose of security, authentication and integrity for a message in network. The paper discusses the algorithm for the function design, its security aspects and implementation details.

Keywords
Authentication, Integrity, Key, MAC, MD5, Security.

1. INTRODUCTION
Hash Function is a function that takes an input of arbitrary length and produces output small but fixed length. Here, usually always the input length is greater than output length. It is a technique for message integrity and if keys are used in the process, it also provides source authentication. Integrity is the technique to transmit the message to receiver without any modification or change in it. Source Authentication provides protection for messages against impersonating and tempering by an active deceiver. G.J. Simmons proposed a model for authentication [1]. The model assumes three active participants- (1) a sender, (ii) a receiver and (iii) an intruder. Here, the intruder impersonates the sender and sends a fraudulent message to the receiver or changes the original message sent by sender.

Use of keyed hash function may solve this issue to the vast extent. As, the key is known to only sender and receiver, and it is being used for authentication or hash generation, then without knowing the actual key, the intruder or deceiver cannot create the new message or change the original one. Thus only sender and receiver may communication using this way. Use of key for hash generation is known as message authentication code (MAC). Moreover, the hash functions, that are used for the purpose of cryptography, in network security, are referred to as cryptographic hash functions specifically.

The recent attacks on MD4 [2], MD5 [3], SHA-0 [4] and SHA-1 [5] by Wang et.al have enforced research in designing new cryptographic hash functions and cryptanalysis of existing ones. [6]. This paper describes the design of a new hash function algorithm with integration of a key. It serves the requirements of message integrity and source authentication both. The proposed algorithm offers features of simplicity as well as speed while implementing on processors of different bits.

2. KEY CONSIDERATION FOR A SECURED HASH FUNCTION
Cryptographic hash functions not only produce a fixed size output from an arbitrarily long input but, for each different input, the output should also be different. Any acceptable cryptographic hash function should possess following three properties [7]:

(a) Pre-Image Resistance- This is also known as one way property. A message $\{0,1\}^* \rightarrow \{0,1\}m$ is pre-image resistant if from given hash value $d \in \{0,1\}^*$ it is impossible to find a message $M \in \{0,1\}^*$ such that $h(M) = d$ i.e. from given hash value, it should be practically impossible to find original message.

(b) Second Pre-Image Resistance- A hash function $h: \{0,1\}^* \rightarrow \{0,1\}m$ is called second pre-image resistant if given a message $M1 \in \{0,1\}^*$ it is impossible to find another message $M2 \in \{0,1\}^*$ such that $h(M1)= h(M2)$. i.e. there should exist no two different messages for which final hash value is same.

(c) Collision Resistance- a hash function $h: \{0,1\}^* \rightarrow \{0,1\}m$ is called collision resistant if it is impossible to find two messages M1 and M2 $\in \{0,1\}^*$ such that $h(M1)= h(M2)$. i.e. M1 and M2 colloid.

Here, in second pre-image resistance, either M1 or M2 is fixed, but in collision resistance both M1 and M2 can be chosen arbitrarily. Thus, a hash function that is collision resistant is always second pre-image resistance also, but reverse is not always true. Apart from these three properties, the hash function should be computationally feasible. But for increased security, sometimes speed of computation and execution is compromised.

3. DESIGN SPECIFICATIONS FOR CRYPTOGRAPHIC HASH FUNCTIONS
Any cryptographic hash function should first of all withstand all the different possible attacks on it, and at the same time, it should also satisfy the requirements as stated in previous section of the paper. As we have already discussed, the input length is arbitrary ($\{0, 1\}^*$ leading to infinite number of inputs) and output is of fixed length ($\{0,1\}m$ leading to few finite number of inputs). Thus due to mapping from infinite to finite length, collision always exists in hash functions. Thus, we may modify our definition of requirement from "it is impossible to find two different messages that produce same hash value" to "it should be very difficult to find two different messages that produce same hash value". This difficulty should be imposed by underlying design algorithm. First of all, Yuval [8] discussed method of finding collisions in hash functions using the Birthday Paradox, which lead to the birthday attack. In this attack, a collision is found with probability $q2 / 2n$ after q queries to a hash function whose output is of n-bit length [9]. Apart from this, to make algorithm work fast, it must include simple operations, such as addition, XOR, complements etc. Furthermore, the security of designed algorithm needs to be proved. Only assumptions of security may lead to failure. And it should also be a modifiable structure so that it may be modified

and made secure against the attacks that will be discovered in future.

4. RELATED WORK

Almost all cryptographic hash functions are based on Merkle-Damgard construction [10]. He proposed few steps for a general purpose hash function generation. Those were padding, append length, initialization of buffer, processing of message in blocks. We may depict those steps as follows:

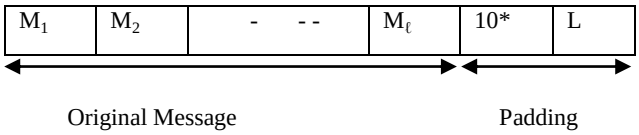

Figure-1 Merkle-Damgard Padding Step

Algorithm- Pads (M)

$D = M+1+64 \bmod m$

$M||1||o^d||<M>64 \rightarrow M^{\wedge}$

$M^{\wedge} \rightarrow M_1 - - - M_\ell$

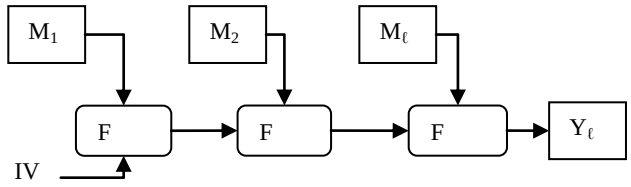

Figure 2: The Merkle-Damgard Construction

Algorithm MD^F:

$M^{\wedge} \rightarrow M_1 - - - M_\ell$
$Y_0 = IV$
for i = 1 to ℓ do
$y_i = F(Mi, y_{i-1})$
return Y_ℓ

Deploying a new hash function includes two constructs- a compression function that operates on input strings of a fixed length and then to use the cascade function to extend the compression function to string of arbitrary length[11]. To improve security aspect with hash function, a key may be used. For this purpose two solutions were proposed. First is Dedicated-Key setting [12], in which a publicly keyed compression function $h: \{0,1\}^k \times \{0,1\}^n \times \{0,1\}^m \rightarrow \{0,1\}^n$ is accessed by a family of hash functions $C^h: K \times M \rightarrow \{0,1\}^n$, such that C^h members are indexed by different public keys $k_i \in K$. This approach facilitates multiple instances of same hash function with multiple keys. And if an attack is found on any particular instance of hash function family, which is indexed by a particular key, it still guarantees of safety of other instances of hash function family, that are indexed by other keys. The only drawback of this approach is need for extra input, in terms of key, resulting in extra calculations and thus, more time, but for increased level of security, this extra time can be afforded [13].

Another approach is Integrated –Key setting [8], which overcomes an important drawback of dedicated-key function, that is: not easily accommodation of key input by keyless compression function. Here, we may take an approach of processing the key only at last compression call, i.e. no need of modifying the compression function, but last hash value will be produced by application of a key too. BCM (Backward Chaining Mode) [14] is a method of construction of hash families without keying all compression function applications. EMD (Enveloped Merkle Damgard) [15] is another same kind of technique. One more variant RMX [16] combines a random salt with every message block before sending it to compression function. This technique makes it suitable for Digital Signature.

5. DESIGN OF PROPOSED ALGORITHM

Typically any hash function has two components: a compression function and a construction. The compression function is a mapping function that transforms a larger arbitrary-size input to a smaller fixed-size output, and the construction is the method by which the compression function is being repeatedly called to process a variable-length message [17]. Traditionally hash functions are being designed without any usage of key component. However, many a few recent attacks have been successfully implemented on these traditional popular hash functions such as- SHA1, MD5 etc. [18, 5, 19]. As we discussed in previous section, security of algorithm needs to be proved, most of the newly designed algorithms are based on previously established and accepted designs with few modifications. If established design promises few security aspects, the new design will automatically do so. In the same line, this algorithm is also based on popular MD5 [20] design. The security notions are assumed from MD5 construction. Furthermore, integration of key in each round of operation on individual blocks gives more strength to the proposed algorithm against many of the known attacks on MD5.

Let us assume an input message M of length b bits. We will use following notations in the description of algorithms:

+ : addition modulo 2^{32}

<<< S: circular left shift by s bit positions

$\wedge$: bit-wise AND

$\vee$: Bit-wise OR

$\oplus$: bit-wise XOR

$\neg$: bit-wise complement

The proposed algorithm may be divided into two phases- preprocessing and hash calculation. The preprocessing phase is very much similar to that of MD-5 and SHA-1, involving padding and message length and further obtaining in m-blocks, each block of 512 bit length. The hash calculation is done on each 512 bit block in iterative manner in second phase of the algorithm. This phase also makes use of two 64 bit keys. The 512 bits are then compressed into 128 bits and provided as input for processing of next block of message. The output of processing of last block of message is called as digest or hash value. The compression function makes use of S-Box, XOR, addition modulo 2^{32} and look-up tables. The use of primitive logical functions, which are implemented on hardware and readily available look-up table help in increasing speed of hash function processing. Following are the few steps of proposed algorithm:

Step 1: Padding-

The original message is padded so that the length of message after padding is congruent to 448 modulo 512 (length ≡ 448 mod 512) this purpose, first bit is always 1 and remaining bits are always 0. This is a compulsory step so, 1 to 512 bits may be appended, depending upon the length of original message.

Step 2: Append Message Length-

After padding, length of original message is appended to the result of step 1. This length is in 64 bit representation. After this step, the length of message is now in multiples of 512.

Step 3: Initialize Buffer-

The algorithm uses a 128 bit buffer (4 words A, B, C and D, 32 bit each), which is initialized with following hexadecimal values:

A = 0 1 2 3 4 5 6 7
B = 8 9 A B C D E F
C = F E D C B A 9 8
D = 7 6 5 4 3 2 1 0
This step is done only for once, and then after receiving the output from first block acts as buffer for second block and so on. The final result of hashing is also stored in this.

Step 4: Initialize t-table-

A 64 element t-table is used in the algorithm, which is prepared by following formula for each t value (ranging from 0 to 63):

$K_t = \lfloor 2^{32} \mid \sin(t+1) \rfloor \mid$ where, t is in radians.

Step 5: Four Secondary Functions-

The algorithm also makes use of four secondary functions f1, f2, f3 and f4, which produce 32 bit word from 32 bit input word. The functions take 16 values from the previously discussed t-table-

$f_1(B,C,D) = (B \wedge C) \vee (\neg B \wedge D)$ for t = 0 , . . . , 15

$f_2(B,C,D) = (B \wedge D) \vee (C \wedge \neg D)$ for t = 16 , . . . , 31

$f_3(B,C,D) = (B \oplus C \oplus D)$ for t = 32 , . . . , 47

$f_4(B,C,D) = C \oplus (B \vee \neg D)$ for t = 48 , . . . , 63

Step 6: Order of words for processing:

The processing is done in 4 rounds. In each round, following sequence of words is used for processing.
Round1: $(j_0,\ldots, j_{15})$ =
(0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15)
Round2: $(j_{16},\ldots, j_{31})$ =
(1,6,11,0,5,10,15,4,9,14,3,8,13,2,7,12)
Round1: $(j_{32},\ldots, j_{47})$ =
(5,8,11,14,1,4,7,10,13,0,3,6,9,12,15,2)
Round1: $(j_{48},\ldots, j_{63})$ =
(0,7,14,5,12,3,10,1,8,15,6,13,4,11,2,9)
Processing is done in blocks. Each block is of 512 bit in length. A word is of 32 bits, thus, each block is made up of 16 words (32 × 16 = 512).

Step 7: Shifting-

Shifting is done in following amounts:
Round1: $(s_0,\ldots,s_{15})$ =
(7,12,17,22,7,12,17,22,7,12,17,22,7,12,17,22)
Round2: $(s_{16},\ldots, s_{31})$ =
(5,9,14,20,5,9,14,20,5,9,14,20,5,9,14,20)
Round1: $(s_{32},\ldots, s_{47})$ =
(4,11,16,23, 4,11,16,23, 4,11,16,23, 4,11,16,23)
Round1: $(s_{48},\ldots, s_{63})$ =
(6,10,15,21, 6,10,15,21, 6,10,15,21, 6,10,15,21)

Step 8: Processing of message in sixteen 32-bit word (512 bit) blocks-

(a) for I = 0 to n-1 do (here, n= number of blocks)
(b) divide M_i into words $W_0, \ldots, W_{15}$ where W_0 is left most word.
(c) Initialization of 4 words ABCD. Here each word is of 32 bit, i.e. total length = 32 × 4 = 128 bit.
A' = A
B' = B
C' = C
D' = D (' represents complement)
(d) For t = 0 to 63 do
$X = B + ((A + f_t(B,C,D) + W_{jt} + K_t) <<< S_t$
A=D
D=C
C=B
B=X
/* end of loop in step d*/
(e) Increment of 4 words ABCD
A = A' + A
B = B' + B
C = C' + C
D = D' + D
(f) Make two 64 bit blocks Y and Z from ABCD
Y = BA
Z = CD
(g) Generate 64 bit key for internal keyed operation. Out of these 64 bits, 8 are used as parity bits and rest 56 bits are used as effective key. Out of this one 56 bit key, 18 keys are generated, each of 48 bit long.
(h) Operations on Y and Z blocks- Both Y and Z are treated similarly. Each block is further subdivided into two partitions- left half of Y block (Ly) and right half of Y block (Ry), and left half of Z block (Lz) and right half of Z block (Rz). Initially the right and left halves (R and L) are permuted (swapped), i.e.
X = L
L = R
R = X
Now next L' and R' are produced as follows-
L' = L
$R' = L (+) f(R_{n-1}, K_n)$
Here, (=) is addition modulo 2^{32}.

This process is repeated for 16 times, each time with a different 48 bit key K.
Thus, $L_n = R_{n-1}$
$R_n = L_{n-1} (+) f(R_{n-1}, K_n)$
After sixteenth round of operation, again perform final permutation (swapping of left and right half), thus,
$X = L_n$
$L_n = R_n$
$R_n = X$
(here X is a 32 bit block used for permutation only.)
Final $X = X \text{ XOR } K_{17}$
Final $Y = Y \text{ XOR } K_{18}$

The algorithm for function f(R.K) is defined as follows-
X= E(R), applying expansion permutation and returning 48-bit data
X' = X ^ k, XOR with the round key
X" = s(X'), applying S boxes function and returning 32-bit data

R' = P(X"), applying the round permutation

(i) Combine final 32 bit values of X and Y. After combining two 64 bit blocks, Y and Z respectively, we get one 128 bit block. These 128 bits are again stored in four 32 bit words ABCD.

/* end of loop in step a. */

(j) After processing last 512 bit block, the final hash value is in ABCD, i.e. output is always 128 bit long digest.

6. SECURITY OF PROPOSED DESIGN

Each individual component of the proposed algorithm has its respective security criteria, that ensures us that the algorithm is secure and collision free. We may give few arguments for security of the algorithm:

- Mathematically secondary functions f1, f2, f3 and f4 are non-invertible and non-linear.
- If individual bits of B,C and D are independent of each other, then overall bits of f(B,C,D) are also independent of each other. It guarantees one-way property.
- Each round access input words in different sequences.
- In fixed point attacks, the attacker tries to produce second pre-image or collisions by insertion of extra blocks into the input [21]. To restrict this attack, padding is done by 1 and number of zeros in preprocessing phase.
- Each step takes input from the output of previous step. Thus, changes done at any one place in any one of the blocks, will surely affect the final output of algorithm. Thus, no two different messages will result in same output hence, proves second pre-image resistance.
- Algorithm works on basic functions, such as modular arithmetic, XOR, addition, left shift, right shift, simple permutation etc. Thus, it does not lead to increased time requirement for processing.
- t-table and all 48 keys can be generated well in advance, so function need not wait in between for table element generation or key generation. This also helps in better execution speed.
- Use of XOR makes sure that output depends on all bits, rather than on neighboring ones.

7. KEY GENERATION AND USAGE

As we have discussed both Dedicated-Key setting and Integrated-Key setting, both use fixed keys, i.e. once a key is dedicated, it will be used for each iteration of compression function. But, in the proposed solution, we will use 16 different key combinations in an iteration of compression function, individually on two word combinations Y and Z respectively. Obviously this approach is more time consuming than keyless one, and it also increases overhead for computing hash by at least n * 2t, where n is total number of blocks and t is computation cost for one block either Y or Z. If we run it in parallel for both of these blocks simultaneously than computation time will increase by only n * t. Now, the efficiency lies in implementation of key function in hashing, and as we have already discussed earlier, because of simpler functions it has come out as a light weight function and will not take much time or efforts for whole message length.

8. IMPLEMENTATION OF THE PROPOSED ALGORITHM

The proposed algorithm can be implemented efficiently on different platforms. It does not require large space (for tables, codes, variables etc.) if large cache memory is used, then higher performance and better throughput can be achieved. This new hash function design uses the same building blocks as MD5 and DES, so we can expect similar performance and space characteristics. But compared to MD5, SHA-0, SHA-1 etc., it provides more security by use of key. Thus, it is a better message authentication code, which may take few seconds, more than MD5 or SHA-1 but at the same time is stronger and less vulnerable.

9. PERFORMANCE ANALYSIS OF PROPOSED ALGORITHM

We have run and tested the function for a large number of inputs. Each time a different key is generated and results in different hash value even for the same input. We tested the algorithm on number of inputs, where input is fixed to a definite size (say 1000 Bytes). In such a case, the execution time is almost same as shown in Figure-3, but each time key is not the same. Different key is being generated even for the same input, thus, it results in different hash value each time. (refer to figure-6 for sample). Similarly, we run this function for different input size (similar number of test cases for same input size between 1000 bytes to 50000 bytes), that is shown in Figure-4. It is found that average execution time is proportional to input size (Figure-5). But as this function is based on predefined MD5 algorithm, there exist no method for getting original message from hash value. And as we have used the concept of key for generating hash value, there is no chance for adversary to compute hash value for a new message and to send it to receiver for the purpose of forging, because we assume that key is known to receiver and sender only.

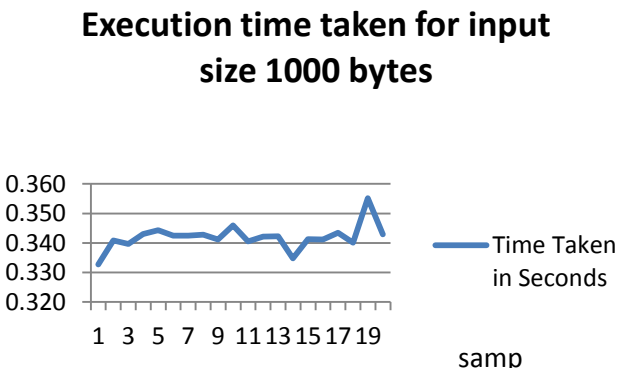

Figure-3: Execution time taken by proposed design for different input test data (of 1000 byte each)

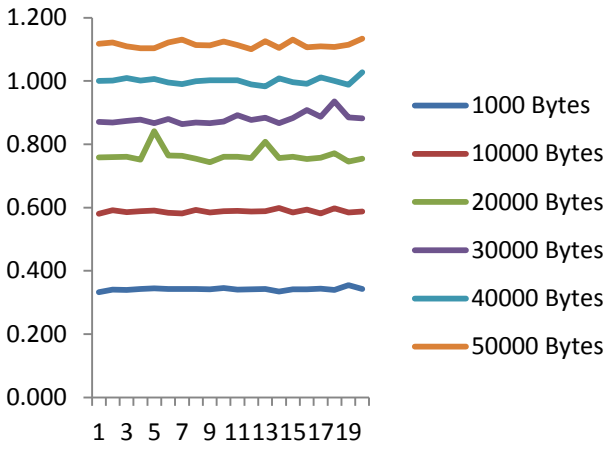

Figure-4: Execution Time (in Seconds) taken by proposed design for input test data of different sizes. (The X axis shows different input samples of same size and Y axis shows execution time for these inputs)

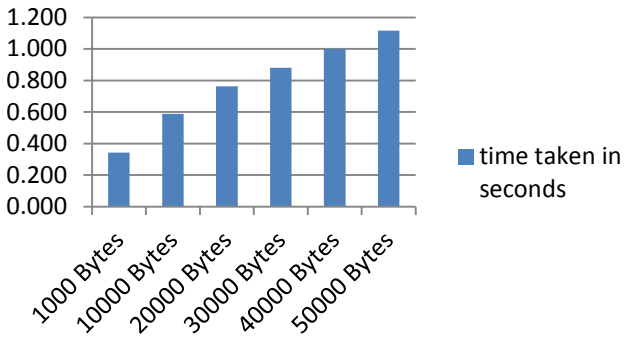

Figure-5: Average Execution Time (in Seconds) for inputs of different size

```
Data length is 1000
Encrypted key:
 42115fbe8833551b4a0b9773fb3b1e8c
Complutation took 0.332673244secs
Data length is 1000
Encrypted key:
 7535cb1da74f78b5792351a924089b68
Complutation took 0.340871827secs
Data length is 1000
Encrypted key:
 fc70acb34cd768f2611f39d9e2651633
Complutation took 0.339580214secs
Data length is 1000
Encrypted key:
 6f96fc19dd5c94fe0c4f71a14b63af30
Complutation took 0.343019806secs
```

Figure 6: Execution of 1000 input byte data (notice different key and varying computation time for each individual input of same size)

Software implementation of the algorithm was tested on system with Intel based CPUs Pentium® -4 2.66 GHz with 1GB RAM. The comparison is given in the following table for various hash functions tested on 1 Mb data file. It shows that the algorithm is the third fastest output after MD5 and RIPEMD. And we may argue in support of lesser speed as compared to MD5 and RIPEMD with the fact that it is using a key in the algorithm and thus providing more security as compared to these algorithms.

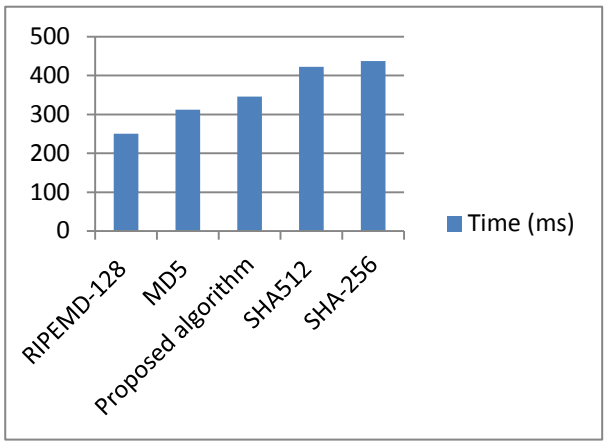

Figure 7: Comparison of execution time taken by few existing hash functions and proposed design

10. CONCLUSION

In this paper we proposed a new hash function algorithm that includes a 64 bit key as an ingredient to the function. It produces 128 bit digest with a secure and simpler technique as compared to many of the popular existing techniques. Use of key adds the source integration facility while creating digest just for integrity purpose. The function has been verified and found fast by using existing tables for number of keys and S-Box.

11. REFERENCES

[1] Simmons GJ. Message Authentication with arbitration of transmitter/receiver disputes. Advances in Cryptology-Eurocrypt'87, Lecture Notes in Computer Science, Springer-Verlag, Berlin; 1988; 304: 151-165.

[2] Wang X, Feng D, Lai X, Chen H and Yu X. Cryptanalysis of the hash functions MD4 and RIPEMD. In Eurocrypt'05, LNCS Springer-Verlag 2005; 3494:1-18.

[3] Klima V. Finding MD5 Collisions on a notebook PC-using multi message modifications. Cryptology ePrint Archive, Report 2005. http://eprint.iacr.org/102.pdf.

[4] Wang X, Yu H, Yin Y. L. Efficient Collision Search Attacks on SHA-0. In Crypto 2005; LNCS 3621, 1-16.

[5] Wang , Yin YL, Yu H. Finding Collisions in the Full SHA-1. In Crypto'05, LNCS Springer-Verlag 2005; 3621:17-36.

[6] Shakeel N, Murtzaa G, Ikram N. MAYHAM- A New Hash Function. International Journal of Network Security, 2011; 15(6): 417-425.

[7] Massierer M. Provably Secure Cryptographic Hash Function. Ph.D. Thesis, School of Mathematics, The University of New South Wales, submitted on December 2006.

[8] Mohammed S A. Al-Kuwari. Integrated-Key Cryptographic Hash Function. Ph. Thesis submitted to University of Bath, Department of Computer Science, September 2011.

[9] Bellare M, Tadayoshi. Hash Function Balance and its Impact on Birthday Attacks. Eurocrypt '04, LNCS Springer-Verlag 2004 ; 3027: 401- 418..

[10] Damgard I. A Design Principle for Hash Functions. Crypto'89, LNCS Springer Verlag 1989; 435 : 416-427, , 1989.

[11] Walker J, Kounavis M, Gueron S, Graunke G. Recent Contribution to Cryptographic Hash Function, Intel Technology Journal 2009: 13 (2): 80-95.

[12] Bellare M, Ristenpart T. Hash Functions in Dedicated Key Settings: Design Choices and MPP Transforms. ICALP'07, LNCS Springer-Verlag 2007; 4596: 399-410.

[13] Rogaway P, Steinberger J. Constructing Cryptographic Hash Function from Fixed-Key Blockciphers. Crypto'08. LNCS Springer-Verlag 2008; 5157: 433-450.

[14] Endreeva E, Preneel B. A Three-Property-Secure Hash Function. SAC '09, LNCS Springer-Verlag 2009; 5381: 228-244.

[15] Bellare, Ristenpart T. Multiproperty- Preserving Hash Domain Extension and the EMD Transform. Asiacrypt '06, LNCS Springer-Verlag 2006; 4284 : 299-314.

[16] Halevi S, Krawczyk H. The RMX transform and Digital Signatures. 2nd NIST Hash Workshop, 2006.

[17] S. Al-Kuwari. Engineering Aspects of Hash Functions. In International Conference on Security and Management (SAM '11), 2011.

[18] Wang X, Feng D, Lai X, Yu H. Collisions for hash functions MD4, MD5, HAVAL-128 and RIPEMD. Cryptology ePrint Archive, Report 2004/1999, 2004.

[19] Wang X, Yu H.. How to Break MD5 and Other Hash Functions. In Eurocrypt'05, LNCS Springer-Verlag 2005; 3494:19-35.

[20] Public-Key Cryptography Standards (PKCS): PKCS #7: Cryptographic Message Syntax Standard: 3.6 Other Cryptographic Techniques: 3.6.6 What are MD2, MD4, and MD5?. RSA Laboratories. Retrieved 2012-10-03.

[21] Rompay B V. Analysis and Design of Cryptographic Hash Function, MAC Algorithms and Block Ciphers. Thesis, Katholieke University Leuven, 2004.

[22] Mornov I. Hash Functions: Theory, Attacks and Applications. Microsoft Research, 2005.

[23] Hirose S, Park JH, Yun A. A Simple Variant of the Merkle-Damgard Scheme with a Permutation. Asiacrypt '08, LNCS Springer-Verlag 2008; 4833 : 113-129.

[24] Tirtea R. Cryptographic hash functions, trends and challenges. Journal of Computer and System Sciences, 2009; 2: 62-65.