



**TRIBHUVAN UNIVERSITY**  
**INSTITUTE OF ENGINEERING**  
**PULCHOWK CAMPUS**

**THESIS NO: 071MSCS658**

**Enhancing MD5 Hash Algorithm using Symmetric Key Encryption**

**by**

**Nitesh Karna**

**A THESIS**

**SUBMITTED TO THE DEPARTMENT OF ELECTRONICS AND COMPUTER  
ENGINEERING IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR  
THE DEGREE OF MASTER OF SCIENCE IN COMPUTER SYSTEM AND  
KNOWLEDGE ENGINEERING**

**DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING**

**LALITPUR, NEPAL**

**MAY, 2017**



**TRIBHUVAN UNIVERSITY  
INSTITUTE OF ENGINEERING  
PULCHOWK CAMPUS**

**THESIS NO: 071MSCS658**

**Enhancing MD5 Hash Algorithm using Symmetric Key Encryption**

**by**

**Nitesh Karna**

**A THESIS**

**SUBMITTED TO THE DEPARTMENT OF ELECTRONICS AND  
COMPUTER ENGINEERING IN PARTIAL FULFILLMENT OF THE  
REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE IN  
COMPUTER SYSTEM AND KNOWLEDGE ENGINEERING**

**DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING**

**LALITPUR, NEPAL**

**MAY, 2017**

## **COPYRIGHT ©**

The author has agreed that the library, Department of Electronics and Computer Engineering, Institute of Engineering, Pulchowk Campus, may make this thesis freely available for inspection. Moreover the author has agreed that the permission for extensive copying of this thesis work for scholarly purpose may be granted by the professor(s), who supervised the thesis work recorded herein or, in their absence, by the Head of the Department, wherein this thesis was done. It is understood that the recognition will be given to the author of this thesis and to the Department of Electronics and Computer Engineering, Pulchowk Campus in any use of the material of this thesis. Copying of publication or other use of this thesis for financial gain without approval of the Department of Electronics and Computer Engineering, Institute of Engineering, Pulchowk Campus and author's written permission is prohibited.

Request for permission to copy or to make any use of the material in this thesis in whole or part should be addressed to:

Head

Department of Electronics and Computer Engineering

Institute of Engineering, Pulchowk Campus

Pulchowk, Lalitpur, Nepal

**TRIBHUVAN UNIVERSITY**  
**INSTITUTE OF ENGINEERING**  
**PULCHOWK CAMPUS, PULCHOWK**  
**DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING**

The undersigned certify that they have read, and recommended to the Institute of Engineering for acceptance, a project report entitled "Enhancing MD5 Hash Using Symmetric Key Encryption" submitted by Nitesh Karna in partial fulfillment of the requirements for the degree of Master of Science in Electronics and Computer Engineering.

.....  
**Supervisor: Dr. Aman Shakya**

Assistant Professor

Department of Electronics and Computer Engineering.

.....  
**External Examiner: Dr. Manish Pokhrel**

Kathmandu University

.....  
**Committee Chairperson: Prof. Dr. Subarna Shakya**

Professor

Department of Electronics and Computer Engineering

.....  
Date

## **DEPARTMENTAL ACCEPTANCE**

The thesis entitled “**Enhancing MD5 Hash Algorithm using Symmetric Key Encryption**”, submitted by **Nitesh Karna** in partial fulfillment of the requirement for the award of the degree of “**Master of Science in Computer System and Knowledge Engineering**” has been accepted as a bonafide record of work independently carried out by him in the department.

-----  
**Dr. Dibakar Raj Pant**

Head of the Department

Department of Electronics and Computer Engineering,

Pulchowk Campus,

Institute of Engineering,

Tribhuvan University,

Nepal.

## **ACKNOWLEDGEMENT**

First of all, i would like to extend my sincere gratitude to Dr. Aman Shakya, the coordinator of Master in Computer System and Knowledge Engineering who kindly contributed his supervision in carrying out my thesis. Without his valuable guidance, comments and encouragement, this study would not have come successfully to this stage.

I am also greatly thankful to all the faculty members and my class mates for their full fledged support.

I am also very grateful to Mr. Kumar Pudasaini for his guidance and supervision time to time.

I would like to thank Mr. Amit Karna, Dinesh Nepal, Suresh Pokhrel and all my friends for their continuous and valuable advice, technical support and their consistent encouragement and inspiration.

I am indebted to my parents for their love, support and blessings. It is their belief in me that kept me going.

## ABSTRACT

Traditionally, hash functions were designed in the keyless manner, where a hash function accepts a variable length input message and produces a fixed length digest as output. However, over the years, few significant weaknesses were found in some popular keyless hash functions. Many a few recent attacks have been successfully implemented on these traditional popular hash functions such as- SHA-1, MD5 etc.

The proposed solution uses integration of keyed symmetric key block encryption algorithm in each step or round of hash compression function. Because symmetric key encryption algorithm works on use of single key, both sender and receiver use the same key.

When comparing MD5 hash and proposed algorithm, the time taken by proposed algorithm raises by 15-20% for data below 50 KB while the number of operation to perform the brute force pre-image attack , second pre-image attack increases from  $2^{64}$  (for MD5) to  $2^{192}$ .

**Keywords:** MD5, DES, Cryptographic hash functions

## TABLE OF CONTENTS

<b>COPYRIGHT ©</b> .....	<b>ii</b>
<b>RECOMMENDATION</b> .....	<b>iii</b>
<b>DEPARTMENTAL ACCEPTANCE</b> .....	<b>iv</b>
<b>ACKNOWLEDGEMENT</b> .....	<b>v</b>
<b>ABSTRACT</b> .....	<b>vi</b>
<b>TABLE OF CONTENTS</b> .....	<b>vii</b>
<b>LIST OF TABLE</b> .....	<b>ix</b>
<b>LIST OF FIGURES</b> .....	<b>x</b>
<b>LIST OF ABBREVIATIONS</b> .....	<b>xi</b>
<b>CHAPTER 1 INTRODUCTION</b> .....	<b>1</b>
1.1 BACKGROUND.....	1
1.2 PROBLEM STATEMENT .....	8
1.3 OBJECTIVES OF THE THESIS .....	8
1.4 SIGNIFICANCE .....	8
1.5 ORGANIZATION OF REPORT .....	9
<b>CHAPTER 2 LITERATURE REVIEW</b> .....	<b>10</b>
2.1 HASH IMPROVEMENT APPROACHES.....	10
2.2 COMPARISON OF ENCRYPTION ALGORITHMS .....	13
2.3 ASYMMETRIC KEY APPROACHES .....	16
<b>CHAPTER 3 METHODOLOGY</b> .....	<b>19</b>
3.1 OVERVIEW.....	19
3.2 FLOW CHART OF THE PROPOSED HASH FUNCTION ALGORITHM .....	22
3.3 PSEUDO CODE OF PROPOSED HASH ALGORITHM.....	22
3.4 DETAIL ANALYSIS OF THE PROPOSED HASH ALGORITHM .....	23
3.5 EXPERIMENTAL SETUP.....	29
<b>CHAPTER 4 RESULT AND ANALYSIS</b> .....	<b>30</b>
4.1 RESULT.....	30
4.2 SECURITY LEVEL ACHIEVED BY THE PROPOSED HASH ALGORITHM	31
4.3 COMPARISON WITH EXISTING APPROACHES.....	33
4.4 ANALYSIS OF TIME REQUIREMENT .....	35



4.5 COMPUTATIONAL COMPLEXITY .....	42
4.6 DISCUSSION .....	45
<b>CHAPTER 5 CONCLUSIONS AND FUTURE WORK .....</b>	<b>47</b>
5.1 CONCLUSIONS.....	47
5.2 LIMITATIONS AND FUTURE WORK.....	47
<b>REFERENCES.....</b>	<b>49</b>
<b>APPENDIX.....</b>	<b>52</b>

## LIST OF TABLE

Table 1: The MD5 Primitive Logic Function .....	6
Table 2: Brief history of attacks on some popular hash functions.....	8
Table 3: Comparison among MD5, SHA1, and MD5 plus . .....	13
Table 4: Comparison of various encryption algorithms .....	14
Table 5: Architecture of the proposed digital signature scheme.....	16
Table 6: Comparison performance of new Digital Signature Scheme .....	16
Table 7: Average Time Required for Exhaustive Key Search.....	17
Table 8: Symmetric key length vs. Brute force combination .....	18
Table 9: Comparison of the Proposed algorithm with the various enhanced approached algorithm on the basis of different parameters. ....	33
Table 10: Properties of existing hash function and Proposed hash function .....	34
Table 11: Time taken by MD5 hash and Proposed algorithm in the range (1KB-50KB) data.....	35
Table 12: Time Taken by Different format (Multimedia, Pdf, jpg) of data by MD5 Hash Algorithm.....	37
Table 13: Time Taken by Different format (Multimedia, Pdf, jpg) of data by Proposed Algorithm.....	39

## LIST OF FIGURES

Figure 1: Simplified Broad Categories of Cryptographic Hash Function .....	3
Figure 2: Simple design of block cipher based hash functions compression function. ....	4
Figure 3: The MD5 Compression Function .....	6
Figure 4: Step Operation of MD5 .....	7
Figure 5: Block diagram of overview of the research methodology .....	19
Figure 6: Flow Chart of Proposed Algorithm .....	22
Figure 7: An illustrated view of processing of proposed hash function- having keyed function in between (HF= hash compression function and EF= keyed function) .....	27
Figure 8: S-Box.....	28
Figure 9: Screenshots of the program output.....	30
Figure 10: Line Diagram of Time taken by MD5 hash and Proposed algorithm in the range (1KB- 50KB) data. ....	36
Figure 11: Bar Diagram of Time taken by MD5 hash and Proposed algorithm in the range (1KB- 50KB) data.....	36
Figure 12: Line Diagram Time Taken by Different format (Multimedia, Pdf, jpg) of data by MD5 Hash Algorithm .....	38
Figure 13: Bar Diagram Time Taken by Different format (Multimedia, Pdf, jpg) of data by MD5 Hash Algorithm .....	38
Figure 14: Line Diagram Time Taken by Different format (Multimedia, Pdf, jpg) of data by Proposed Hash Algorithm.....	40
Figure 15: Bar Diagram Time Taken by Different format (Multimedia, Pdf, jpg) of data by Proposed Hash Algorithm.....	40

## **LIST OF ABBREVIATIONS**

MD5: Message Digest 5

DES: Data Encryption Standard

KDC: Key Distribution Center

# CHAPTER 1

## INTRODUCTION

### 1.1 BACKGROUND

Cryptographic Hash Functions are important tool for message integrity in modern Cryptography. Hash function is a function, whose input is any message and output is a hash-value, hash-result, hash-code, digest or simply hash [2]. Simply, it may be defined as a function  $h$ , which maps bit-strings of arbitrary finite length to  $n$ -bit fixed length string. The output size of hash ranges normally from 128 bit to 512 bits (digest smaller than 128 bits are considered to be insecure and larger than 512 bits would create more overhead while transmission), depending upon the specific hash function being used. If the hash function gives an output of an  $n$ -bit digest, it is known as  $n$ -bit hash. For a domain (denoted as  $Do$ ) and range (denoted as  $Ra$ ) with function value  $HASH: Do \rightarrow Ra$  and  $|Do| > |Ra|$ , the function is many-to-one. It implies that we may not avoid identical output as hash value for dissimilar pairs of inputs. In fact, if we may restrict  $HASH$  to a  $m$ -bit input domain where,  $t > n$ , and  $HASH$  is supposed to be “random” considering all outputs to be essentially equi-probable, then about  $2^{m-n}$  inputs would map to every output, and probability of two randomly chosen inputs to colloid to the same output is  $2^{-n}$  (independent of  $m$ ). The fundamental concept behind hash functions in cryptography is that a digest is treated as a small representative image of given input string which can be used with an assumption that it is uniquely identifiable with that input string. This small representative image is also called an imprint, digest digital finger print, or simply the message digests [4].

These functions make use of a key in the process of generating a hash value. Therefore, these functions require two specific inputs:

- (1) a message of arbitrary finite-length, and
- (2) a key of specific length.

The fundamental approach behind this is that, if adversary does not know the key, he must not be able to forge the message. Such type of hash functions are also known as Message Authentication Codes (MAC). Output of MAC depends on both – the message and the key. As general property of any hash function, the output of keyed hash functions is also of pre-specified length.

### **KEYED HASH FUNCTIONS:**

These functions make use of a key in the process of generating a hash value. Therefore, these functions require two specific inputs: (1) a message of arbitrary finite-length, and (2) a key of specific length. The fundamental approach behind this is that, if adversary does not know the key, he must not be able to forge the message. Such type of hash functions are also known as Message Authentication Codes (MAC). Output of MAC depends on both – the message and the key. As general property of any hash function, the output of keyed hash functions is also of pre-specified length.

### **DEFINITION-1(KEYED HASH FUNCTIONS):**

“The map  $HASH : \{0,1\}^* \times \{0,1\}^n \rightarrow \{0,1\}^m$  is said to be a keyed hash function with  $m$  -bit output and  $n$  -bit key if  $H$  is a deterministic function that takes two inputs, the first of an arbitrary length, the second of  $n$  -bit length and outputs a binary string of length  $m$  -bits. Where both  $n, m$  are positive integers.  $\{0,1\}^m$  and  $\{0,1\}^n$  are the sets of all binary strings of length  $m$  and  $n$  respectively and  $\{0,1\}^*$  is a set of all finite binary strings. Keyed hash function or MACs are majorly concerned with message integrity and source authentication both [4].

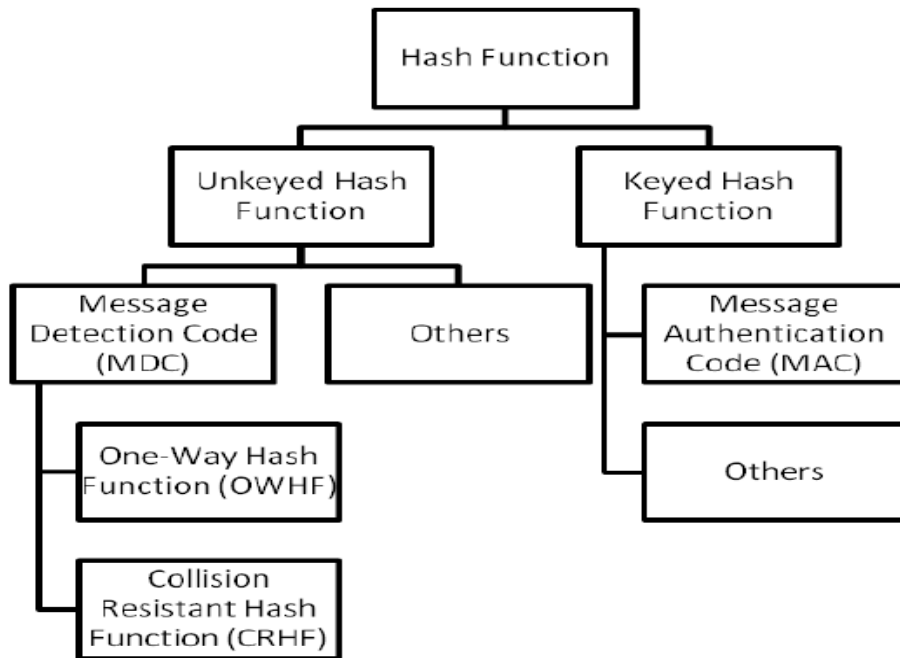
### **UNKEYED HASH FUNCTIONS:**

These are the hash functions that do not use any key as input to generate hash value. Most of the hash functions are unkeyed hash functions. Almost all hash functions that are being used since the early 1990" s in cryptography come under this category. By appending the digest to the message during the transmission, these hash functions are used for error detection. The error can be diagnosed, if the digest of the received message, at the receiving end is not equal to the received message digest. This is also known as Modification Detection and thus such hash functions are also called Manipulation Detection Codes or Modification Detection Codes (MDC) or Message Integrity Codes (MIC). Infact, keyed hash functions can also be used for error detection but the unkeyed hash functions are easier to use for this application because there will not be any problem of secrecy of key used. Unkeyed hash functions are only concerned with message integrity [4].

### **DEFINITIONS-2(UNKEYED HASH FUNCTIONS):**

“The map  $H : \{0,1\}^* \rightarrow \{0,1\}^m$  is said to be an *unkeyed hash function* with  $m$  –bit output if  $H$  is a deterministic function that takes an arbitrary length message as input and

outputs a binary string of length  $m$  -bit. The notations  $m$ ,  $\{0,1\}^m$  and  $\{0,1\}^*$  are similar as that of used in Definition-1.



**Figure 1: Simplified Broad Categories of Cryptographic Hash Function [2].**

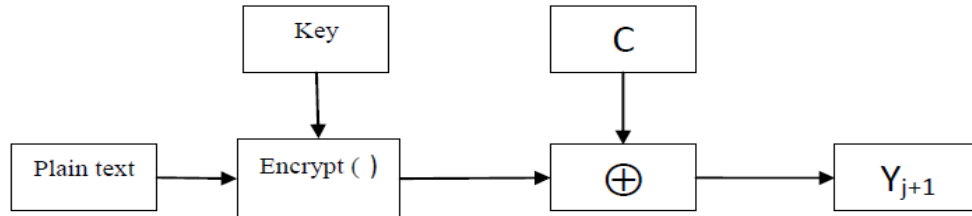
### **HASH FUNCTIONS BASED ON BLOCK CIPHERS:**

As per S.T. Bartkewitz [7], it is always desirable to use the hash functions that are based on already proven and established methods of design. Because block ciphers provide some appropriate properties, they could be used in hash function design. Hash functions may start working upon input of any random length, but they process them in equal sized blocks, so there should be a method to transform these inputs in small and equal length blocks. Here comes the use of block ciphers whose fixed-length input is much smaller. Merkle-Damgard first used this approach in their Merkle-Damgard construction [6].

In comparison of the dedicated hash functions, the hash functions will usually execute consuming more time if they are based on block ciphers. However it is still advisable to use such slower hash functions because they are easy to implement and they can be used for both of the purpose that is for generating block cipher encoded form of plain text and for generating digital signature through hash function. Few of such block

cipher based hash methods are Davies-Meyer, Matyas-Meyer-Oseas, Miyaguchi-Preneel, MDC-2 and MDC-4 compression function.

Following diagram well describes block ciphers based general construction of a compression function for hash functions [6]:



**Figure 2: Simple design of block cipher based hash functions compression function.**

In the above figure 1.3(a) “Encrypt ( )” is a block encryption technique which accepts “Plain text” as input along with a function key “Key”. X is some arbitrary length message which is divided into m number of blocks and it takes one round to process each block. We chose various required values such as the input “Plain text”, the key “Key” and the XOR value “C” from the set  $Z = \{A, X_j, Y_j, X_j \oplus Y_j\}$ , [3]

Where,

A= a constant value

$Y_j$  = the output of the previous round

$X_j$  = the current message block being processed

j = number of message blocks currently being processed.

### **MD5 Hash Algorithm**

The MD-5 message digest algorithm was also designed by Rivest [25]. As other hash functions, it also takes a arbitrary length message as its input and gives a digest having 128-bit length as final output. The complete operation of MD5 hash function is described in the given steps:

#### **Step 1: Append padding bits:**

In first step, the message is appended with few bits to make its length congruent to 448 modulo 512 bit ( $\text{length} \equiv 448 \pmod{512}$ ). It gives the padded message length 64-bits lesser than integer multiple of 512-bits. This is a necessary step and is always executed; no matter the message is itself of the required length. The message M is broken



into equal size partitions, known as blocks  $M = m_1, m_2, \dots, m_n$ . Each block is of  $l$  -bits and there are total  $n$ -number of blocks.

**Step 2: Append Message length:**

A sixty-four-bit representation of the original message length (before padding) is placed after the output of the step-one. In case the length of initial message is more than  $2^{64}$  bits then only the 64-last significant bits of the length are accepted. In this way, the initial message length modulo  $2^{64}$ , is placed in this field.

**Step 3: Initialization of buffer:**

Because the MD5 hash function produces 128-bit long digest, it makes use of a 128-bit buffer to store initial value, intermediate values and final output of the hash operation. The buffer can be represented as 4 thirty two bit words- A, B, C and D. The values of the 128-bit buffer words for MD-5 are presented below (in hexadecimal):

Word A: 67452301

Word B: EFCDAB89

Word C: 98BADCFE

Word D: 10325476

**Step 4: Process the message in blocks with 512-bit length:**

This is the important stage. The message is processed in blocks of 512-bit, separately on each block. The important area of the function is the compression operation, which comprises of 4 processing rounds. Every round works on the present 512-bit long block that being processed and is represented as  $m_j$  where  $j = 1, 2, \dots, n$  and it also accepts ABCD, the 128-bit buffer that is modified within every round. It also makes use of a T-Table that consists of 64 elements  $T [1, 2, \dots, 64]$ . This T table is constructed from the sine function. In each round, one fourth of the T-table is used. The diagrammatic representation this process is shown in Table 1 [25].

All four rounds of MD5 compression function are similar in structure; the only difference is that a different primitive logical function is being used in every round, which is referred to as F for round 1, G for round 2, H for round 3 and I for round 4. It uses three buffer words- B, C and D. The logical operators OR (represented by the symbol  $\vee$ ), AND (represented by the symbol  $\wedge$ ), and XOR (represented by the symbol  $\oplus$ ) are the three functions used in F, G, H and I.

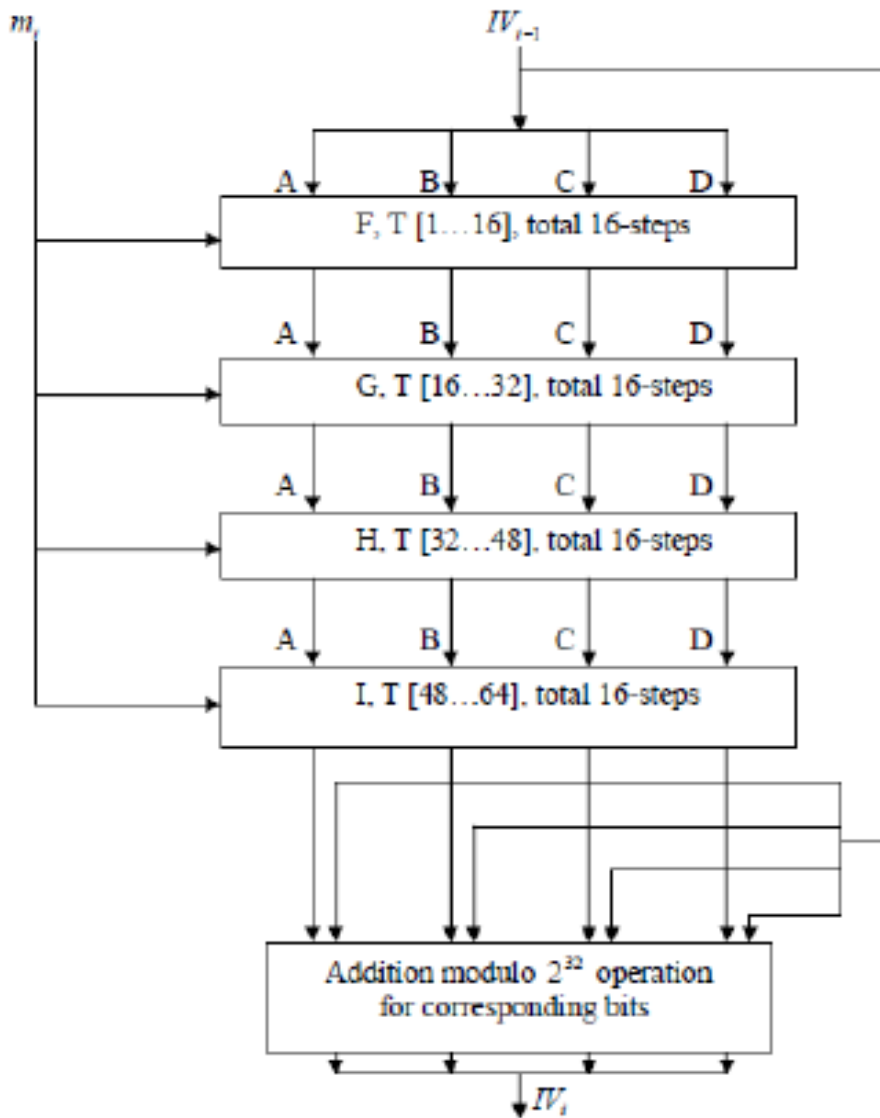
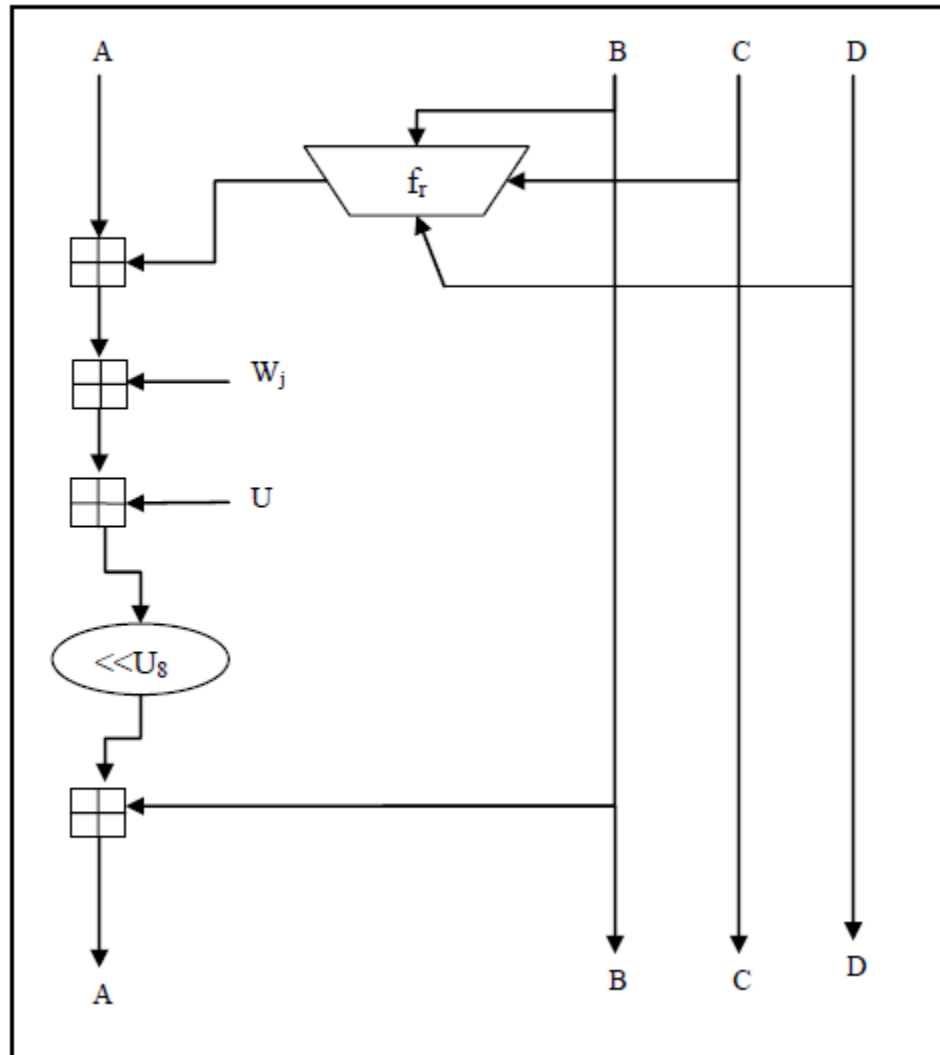


Figure 3: The MD5 Compression Function [25]

Table 1: The MD5 Primitive Logic Function [25]

Round	Primitive Logical Function	$\sim(B \ C \ D)$
1	$F(B, C, D)$	$(B \wedge C) \vee (B \wedge D)$
2	$G(B, C, D)$	$(B \wedge D) \vee (C \wedge D)$
3	$H(B, C, D)$	$B \oplus C \oplus D$
4	$I(B, C, D)$	$C \oplus (B \vee D)$

The output of the last round is then summed up with the input of initial round and the sum operation is performed individually for all 4 words A, B, C and D of the buffer with every respective word given in the input.



**Figure 4:** Step Operation of MD5 [25]

**Step 5: Output:**

When all 512 bit blocks have been processed, the output from the final stage becomes the 128-bit hash value and may be sent to receiver for message integrity.

## **BRIEF HISTORY OF ATTACKS ON SOME POPULAR HASH FUNCTIONS:**

Though, the hash constructions are strong enough to give unique output for each unique input value, as per the definitions of strong and weak collision resistance for hash requirements, still these constructions are not secure from attacks. The first attempts to break the MD-family have targeted the round functions of MD4 and MD5 and were done by Merkle, Den Boer and Bosselaers [9, 10] and by Vaudenay. The table illustrates the important attack approaches on MD-family.

**Table 2: Brief history of attacks on some popular hash functions [8]**

NAME	HASH FUNCTION	COMPLEXITY
Dobbertin	MD4	$2^{20}$
Dobbertin	MD5	About 10 Hours
Chabouad, Joux	SHA0	$2^{61}$
Wang, Yu	MD5	$2^{39}$
Wang, Yin, Yu	SHA0	$2^{39}$
Wang, Yin, Yu	SHA1	$2^{69}$
Wang, Yao, Yao	SHA1	$2^{63}$

### **1.2 PROBLEM STATEMENT**

1. Over the years the traditional keyless hash functions were not found secure. Few significant weaknesses are found in some traditional popular hash function as- MD5, SHA1-etc.
2. What will be the time overhead if we use key to encrypt the keyless hash function so as to make the communication more secure.

### **1.3 OBJECTIVES OF THE THESIS**

1. To enhance existing MD5 Hash algorithm using Symmetric Key encryption.
2. To compare the enhanced algorithm in terms of time complexity and security.

### **1.4 SIGNIFICANCE**

This thesis contributes towards more secure message transfer over internet while securing message integrity and source authenticity both. It clearly states that any attacker requires at least  $2^{192}$  calculations before a successful brute force pre-image and brute force second pre-image attack to perform on Proposed Hash. This is quite a large amount of calculation, which is considered among secure number of calculations (if calculation is

more than  $2^{64}$ , it is considered to be safe) [23], because practically it is not feasible to execute such a large number of test.

The proposed algorithm is best applicable for the low size data as password hashing, Virus checking, Message Authentication Code, Digital Signature, the data of IOT, Sensors data, military low size secret data and other low size related data.

## **1.5 ORGANIZATION OF REPORT**

1. The first chapter discusses on the basic cryptographic hash functions, brief history of attacks on some popular hash functions, problem statement of the thesis, objective of thesis, its scope of study.
2. The second chapter briefly explains summary of the research papers which have been reviewed for the thesis.
3. The third chapter is about the design and analysis of the proposed hash function while using a key. Here the algorithm is designed, and then executed the function for number of input messages. The speed of function is analyzed for different sizes of input, as speed plays an important role in security.
4. The chapter four of the thesis presents the result and analysis of the proposed hash algorithm.
5. Chapter five gives the conclusion, limitation and future work of the thesis.

## **CHAPTER 2 LITERATURE REVIEW**

The review of literature involves the systematic identification and analysis of documents related to the study under taken review of the previous studies helps to conduct the new research in systematic manner by providing the general outline of the research study and avoids the unnecessary duplications.

Realizing the importance of literature review some efforts are made here to present the significant results or conclusions of different studies mainly focusing to the opinions towards improving the existing MD5 hash algorithm. Some works in opinions and the related topics are presented here.

### **2.1 HASH IMPROVEMENT APPROACHES**

Solanki and Agarwal [1] proposed an algorithm to enhance the security and tensile strength of the MD5 algorithm. They described that the improvement was introduced by providing any hash (password + salt + key), that is the system of manipulation takes the user defined password, a developer defined salt and a key calculated from the system's input as the initial inputs. According to them "the security of message digest algorithm has been raised with the help of hashing this algorithm with other mechanism also that is using the salt given from the developer to be implemented with IDEA algorithm and quantifying a value through this procedure". Then the salt which is created as cipher text is now assumed as a qualifier variable that is, it has to be inputted into a quadratic equation which provides two roots. Finally these roots are imposed into the message digest algorithm, as the first root is applied as salt in the expansion phase, while the second root evaluated from the quadratic equation computation is used as the salt in the decompression system phase in the same algorithm. Therefore at the final stage the view ability of the cipher text has changed and thus no existing frame can decrypt the cipher text created with this implementation entire setup from scratch [1].

Sharma and Pathak used a Genetic algorithm with Ring crossover and other operators for the cryptanalysis of Simplified Data Encryption Standard. They found that Genetic algorithm is far better than brute force search algorithm for cryptanalysis of S-

DES. Although S-DES is a simple encryption algorithm, GA with Ring Crossover method can be adopted to handle other complex block ciphers like DES and AES [2].

Boonkrong and Somboonpattanakit [3] suggested that the main problem with the most popular functions for storing passwords is that they were not designed to serve such purpose. An attack using a rainbow table is possible. In order to counter this type of attack, a salt value has been introduced. Their proposed method [3] begins by checking the quality of the original password before randomly generating a salt value suitable for the password. From their experiment they found that in order to withstand a rainbow table attack, the size of the salt value must be between 80 and 256 bits or 10 and 32 characters. The salt value is then inserted into the password according to the placement pattern. Next, this combination of salt value and password is to be hashed by a one-way hash function. The resultant hash value is the value to be stored in the password database. It can be seen that, by using the proposed technique, even if an adversary gets hold of the hash value and the salt, it is very difficult that the password will be compromised. This is because the salt is placed in the password will never be known unless the plaintext password is known [3].

Ogini and Ogwara [4] from their paper concluded that Hashing algorithms such as MD5 used for encrypting plaintext passwords into strings that theoretically cannot be deciphered by hackers due to their one-way encryption feature. However, in order to prevent hackers that used password guess attacks, they introduced the salting pattern such that each password is hashed with a salt which can be in the form of random numbers generated during the process of calculating the hashed function or using some parts of their logon details which is the salt that is not know even to the user password to reduce the incident of being cracked and made stronger compared with that of encryption which is a two way process.

Salih Mohammed [5] introduced and tested an alternative improvement model. Their results shows that the suggested model has the ability to increase the security by about 4% due to the using of key in the same locations of the expanded data in each round. A 12.5% increasing of security was also obtained by using the 64-bit key instead of 56-bit used in the standard model. The blocks of DES model and the proposed model are the same and without addition of new blocks, also it can be switched between the

proposed and the standard model at any round according to a secure key used for switching between models. The last case increased the time required to attack the algorithm because we have  $16!$  case for switching between the two models [5].

Sombir and Sunil [6] said that the original DES implementation has some weaknesses, and to overcome the most of weakness, the Enhanced DES algorithm is designed. The Designed system improved the security power of original DES. The only drawback of Enhanced DES is extra computation is needed but the today's computer have parallel and high speed computation power so the drawback of the Enhanced DES algorithm is neglected because our main aim is to enhance the security of a system. By using the Enhanced DES algorithm the security is very tight and approximately impossible to crack and break the Enhanced DES algorithm [6].

Roshdy and Dahab [7] proposed a new secure hash algorithm based on the previous algorithms, MD5 and SHA-256 that can be used in any message integrity or signing applications where its hash code length is 256 bits. The complexity of the proposed hash algorithm is higher than that of SHA-256 and MD5. The test results of the proposed algorithm show that its security is higher than that of SHA-256 and MD5. The proposed algorithm passes the avalanche test and differential attack test with probability greater than SHA-256. The proposed algorithm is immune to differential attack since the probability of hash value before and after changing bits in previous position is greater than 50% [7].

Priyanka and Vivek [8] implemented a new algorithm from the existing algorithm MD5. There may be any input maximum of 512 bit will convert the encrypted output of 512 bit. The output is encoded from the input and the output would always is of 512bit message. They concluded that there is one more application of this algorithm is Message Authentication Code (MAC). This is an integrity check mechanism based on cryptographic hash functions using a secret key. Typically, message authentication codes are used between two parties that share a secret key in order to validate information transmitted between these parties [8].

Sharma and Sunita [9] implemented a new algorithm for 640 bit message digest. According to the paper this new algorithm provides high security in data transfer. For the implementation of this algorithm they used message digest5 - 128 bit algorithm as a basic



element and create a application or 640 bit messages. The output would always is of 640 bit message. In this way the secret information like passwords can be shared with the peer [9].

Chan and Liu [10] worked on an algorithm that produced 160 bit and that is based on MD5, only introducing one excellent assistant function from 160-bit SHA1. They named it MD5plus algorithm. They concluded that MD5plus algorithm has palpable advantages not only in security, but also in computing time spending. Besides, combining with other 160-bit asymmetric algorithms, we could use MD5plus to compose an excellent plaintext protection system [10].

**Table 3: Comparison among MD5, SHA1, and MD5 plus [10].**

Function	MD5	SHA1	MD5 Plus
Block Length	512 bit	512 bit	512 bit
Algorithm length	128 bit	160 bit	160 bit
Rotation Steps	64 steps	80 steps	80 steps
Initialization Variables	4	5	4
Collision Complexity	$2^{64}$	$2^{80}$	$2^{64} + 2^{80} = 2^{64}$

Singh, Goel [11] often suffer from various security attacks because of its features like open medium, topology dynamically, and management, cooperative. This study has considered the hybrid system using MD5 & RSA encryption algorithm as a means of data security according to attacks. In this method the MD5 & RSA are combined, to improve the security of such network. This secures the data as well as preserves the confidentiality and secure [11].

## 2.2 COMPARISON OF ENCRYPTION ALGORITHMS

Mahajan & Sachdeva [12] shows the comparison between AES, DES and RSA. Based on the text files used and the experimental result they concluded that AES algorithm consumes least encryption and RSA consume longest encryption time. From their simulation result, they evaluated that AES algorithm is much better than DES and RSA algorithm [12].

Xie, Liu, et al [13] in their paper, shows how to choose right input differences for MD5 collision attack, and analyze their complexities and answered Stevens' challenge response for a completely new single block MD5 collision in three ways.

- Firstly, Stevens' single block MD5 collision is not a completely new one, since it can be simply derived from our original one.
- Secondly, Stevens' single block MD5 collision is much more inferior to our original one in computational complexity.
- Thirdly, Stevens had not found the very optimal solution that we preserved and had been wishing that someone could also find it, whose collision complexity is about 241 MD5 compressions [13].

Jasek and Sarga [14], proposed that MD5 have been proven insecure against several attacks under realistic assumptions and its use is therefore discouraged in favor of more resilient, key-stretching iterative hashing algorithms. Despite no SHA-1 hash collisions have been produced so far, advances as per Moore's law, complexity and future-proof modifiability should be taken into account when selecting suitable hashing functions for sensitive data [14].

**Table 4: Comparison of various encryption algorithms [15]**

Factors	AES	3DES	DES
Key length	128,192 or 256 bits	(k1,k2,k3)168 bits (k1, k2 same)112 bits	56 bits
Cipher Type	Symmetric block	Symmetric block	Symmetric block
Block Size	128,192 or 256 bits	64	64
Developed	2000	1978	1977
Security	Consider Secure	one only weak which is Exit in DES	Proven inadequate
Possible Keys	$2^{128}, 2^{192}, 2^{256}$	$2^{168}, 2^{112}$	$2^{56}$
Time Required to check all possible keys at 50 billion keys per second	For 128 bit keys: $5 * 10^{21}$ years	For 112 bit keys: 800 Days	for 56 bit key: 400 Days

Nadeem, Dr Javed [16] has implemented the popular secret key algorithms including DES, 3DES, AES (Rijndael), Blowfish and compared their performance by encrypting input files of varying contents and sizes. According to the paper, the algorithms were implemented in a uniform language, using their standard specifications, and were tested on two different hardware platforms, to compare their performance and concluded that the Blowfish was the fastest algorithm [16].

Singh, and Maini [17] on the other hand provides a performance comparison between four of the most common encryption algorithms: DES, 3DES, Blowfish and AES. The comparison has been conducted by running several encryption settings to process different sizes of data blocks to evaluate the algorithm's encryption/decryption speed. They showed that Blowfish has better performance than other commonly used encryption algorithms. Since Blowfish has not any known security weak points so far, it can be considered as an excellent standard encryption algorithm. AES showed poor performance results compared to other algorithms, since it requires more processing power [17].

Elminaam and Hadhoud [18] presented a performance evaluation of selected symmetric encryption algorithms. The selected algorithms were AES, DES, 3DES, RC6, Blowfish and RC2. They concluded following points as:

There is no significant difference when the results are displayed either in hexadecimal base encoding or in base 64 encoding.

- in the case of changing packet size, it was concluded that Blowfish has better performance than other common encryption algorithms used, followed by RC6.
- in the case of changing data type such as image instead of text, it was found that RC2, RC6 and Blowfish has disadvantage over other algorithms in terms of time consumption.
- 3DES still has low performance compared to algorithm DES.
- in the case of changing key size – it can be seen that higher key size leads to clear change in the battery and time consumption [18].

Seth, Mishra concluded that, based on the text files used and the experimental result it was concluded that DES algorithm consumes least encryption time and AES algorithm

has least memory usage while encryption time difference is very minor in case of AES algorithm and DES algorithm while RSA consume longest encryption time and memory usage is also very high but output byte is least in case of RSA algorithm [19].

### 2.3 ASYMMETRIC KEY APPROACHES

A research paper [20] by Kuppuswamy and Dr. Al-Khalidi presents a new variant of digital signature algorithm which is based on linear block cipher or Hill cipher initiate with Asymmetric algorithm using mod 37. The proposed method of Digital Signature Scheme based on the linear block cipher or Hill cipher. It is basically symmetric key algorithm. But, here they implemented the Symmetric key algorithm as a Asymmetric key algorithm and used in Digital Signature scheme The algorithm executes on PC computer of CPU Intel Pentium 4, 2.2 MHz Dual Core. It is tested with messages and with different length of 100 characters. In Table 4, it has been mentioned clearly architecture of the proposed Digital signature scheme and Table 5 shows the comparison performance of new Digital Signature Scheme [20].

**Table 5: Architecture of the proposed digital signature scheme [20]**

Algorithm	Key Selection Procedure
RSA Digital Signature	Between any two large Prime
Elliptic curve	$Y^2 = x^3 + ax + b$ on real numbers
Proposed Digital signature	Using block cipher symmetric algorithm

**Table 6: Comparison performance of new Digital Signature Scheme [20]**

Algorithm	Number of Characters(Message)	Execution Timing
RSA Digital Signature	100	5.6 seconds
Elliptic curve	100	5.4 seconds
Proposed Digital signature	100	5.2 seconds

Several points that they concluded from their experimental results are:

- The proposed method consumes least encryption time (computing time) and others has taken maximum time in encryption for same amount of the data
- The performance of the proposed algorithm is found to be competitive to the most of the digital signature algorithms which are based on multiple hard problems [20].

Barik and Dr. Karforma [21] proposes a signed transmission scheme using standard RSA Digital Signature with implemented version of MD5 algorithm to ensure Message Integrity, Privacy, Nonrepudiation and Authenticity. In this paper [21], they have proposed asymmetric encryption with Digital Signature to maintain data integrity customize hash function. According to them the result will be more efficient when someone apply dual Digital Signature [21], [22].

A book by William Stallng [23], summarize the time taken to encrypt and decrypt with various sizes of key (bits) as:-

**Table 7: Average Time Required for Exhaustive Key Search [23].**

Key Size (bits)	Number of Alternative Keys	Time Required at 1 Decryption/ $\mu$ s	Time Required at $10^6$ Decryptions/ $\mu$ s
32	$2^{32} = 4.3 * 10^9$	$2^{31} \mu s = 35.8$ minutes	2.15 milliseconds
56	$2^{56} = 7.2 * 10^{16}$	$2^{55} \mu s = 1142$ years	10.01 hours
128	$2^{128} = 3.4 * 10^{38}$	$2^{127} \mu s = 5.4 * 10^{24}$ years	$5.4 * 10^{18}$ years
168	$2^{168} = 3.7 * 10^{50}$	$2^{167} \mu s = 5.9 * 10^{36}$ years	$5.9 * 10^{30}$ years
26 characters (permutation)	$26! = 4 * 10^{26}$	$2 * 10^{26} \mu s = 6.4 * 10^{12}$ years	$6.4 * 10^6$ years

Kasgar and Dhariwal [25] explains that Brute Force Attack or Exhaustive KeySearch being an application of Brute Force Search is a Known Plaintext Attack (KPA) which requires little bit cipher text and corresponding plaintext. In this technique, the correct key is found by checking all possible keys systematically. The practical feasibility factor of this attack depends on the key length used for encryption purpose. The success rate of this attack depends on the number of keys to be tested and the speed of per key test. To make this attack infeasible, key with long key length must be selected. For example, the system which could brute-force a 56- bit encryption key in one second, would take 149.7 trillion years to brute-force a 128-bit encryption key. The following table illustrates the example more clearly-

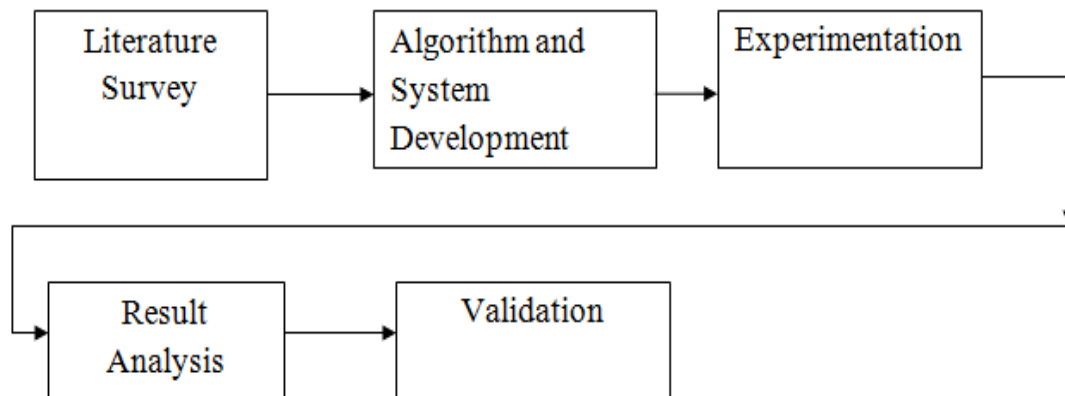


## CHAPTER 3 METHODOLOGY

Currently, there are two methods being used to extend or strengthen the previously existing designs: First, to perform increased number of rounds of operations instead of prescribed number of functions in existing hashing algorithm (for example, instead of four primitive functions use more in case of MD5); or add some advanced coding or permutation steps (for example, use more scrambling techniques in SHA-1); second to increase the total buffer space and use different mixing step in each of the round. Building hash functions using block ciphers, as a base, is the most popular and most widely applied method. Hash functions that use this method, use a compression function that is like a block-cipher consisting of two inputs- a block of message and a key. At present, a protocol is considered strong and secure if it requires at least  $2^{128}$  operations to perform attack on it.

### 3.1 OVERVIEW

The overview of the Research Methodology can be from the block diagram



**Figure 5: Block diagram of overview of the research methodology**

In designing an algorithm, the first and most important step is to learn the previous attempts made in that field. For this thesis, more than 25 research papers have

been reviewed and summarized under various approach as: Hash improvement approach, Encryption algorithms comparisons approach and symmetric key approach.

The main portion of the thesis is the system development. For this stage the main difficulty is to design the algorithm with less complexity which also provides more security than the existing hash function. The detail of the algorithm steps is explained in the next section of the chapter.

For the experiment, large data has been tested. For data less than 50 KB only text type of data is used while for the data size above 100 KB up to 10 MB, various format of data as pdf, jpg and multimedia (mp3, mp4, avi) type of data is used to prove that the proposed algorithm is suitable for the all format of data. The program is executed in Lenovo i5 processor windows 7 with 2GB RAM and 500GB HDD.

The result analysis shows that the time taken by the proposed hash algorithm for less than 50 KB is only 15-20% more than the existing hash algorithm while the number of operations required to perform brute force attack increases from  $2^{64}$  to  $2^{192}$ .

## **PROPOSED HASH ALGORITHM**

The proposed solution uses integration of keyed symmetric key block encryption algorithm in each step or round of hash compression function. Because symmetric key encryption algorithm works on use of single key, both sender and receiver use the same key. This common key may be shared between them using an encrypted link between them by Key Distribution Center (KDC).

It is the responsibility of KDC to send the common session key to both sender and receiver in communication. KDC uses master keys of both parties for this purpose. Because only these two parties carry their corresponding private keys, so no other user in the network may intercept and read the original message and make use of this session key. Except KDC and both parties, involved in the message transmission, no other user has any idea of the shared secret key. Thus, this method helps in validating identity of source as key with sender and receiver is now same. In this solution, the working of hash compression function will be copulated with keyed encryption function. The output of compression function in each block will further used as input for keyed operation. Compression function will give output of 128 bit long and keyed function will take an input block of 64 bit at a time. Thus, first, the output of compression function will be



divided into two equal sized blocks, with a length of 64 bit, then it operates two times: initially, with left 64 bits and then with right 64 bits. Then the keyed function (encryption function) will be applied on both 64 bit blocks one by one. The final output will be of 64 bit for each left and right part, making total of 128 bit. This overall 128 bit output will then be used as 128 bit CV<sub>q</sub> for compression function processing of next block of input [4].

Assign IV to CV<sub>0</sub>

Assign (E (K, B<sub>1</sub>) || E (K, B<sub>2</sub>)) to CV<sub>q</sub>

Here,

IV = MD buffer Initialization value as set by given compression function

E = Block encryption scheme

B<sub>1</sub> = Left 64 bits from output of hash value of 512 bit block

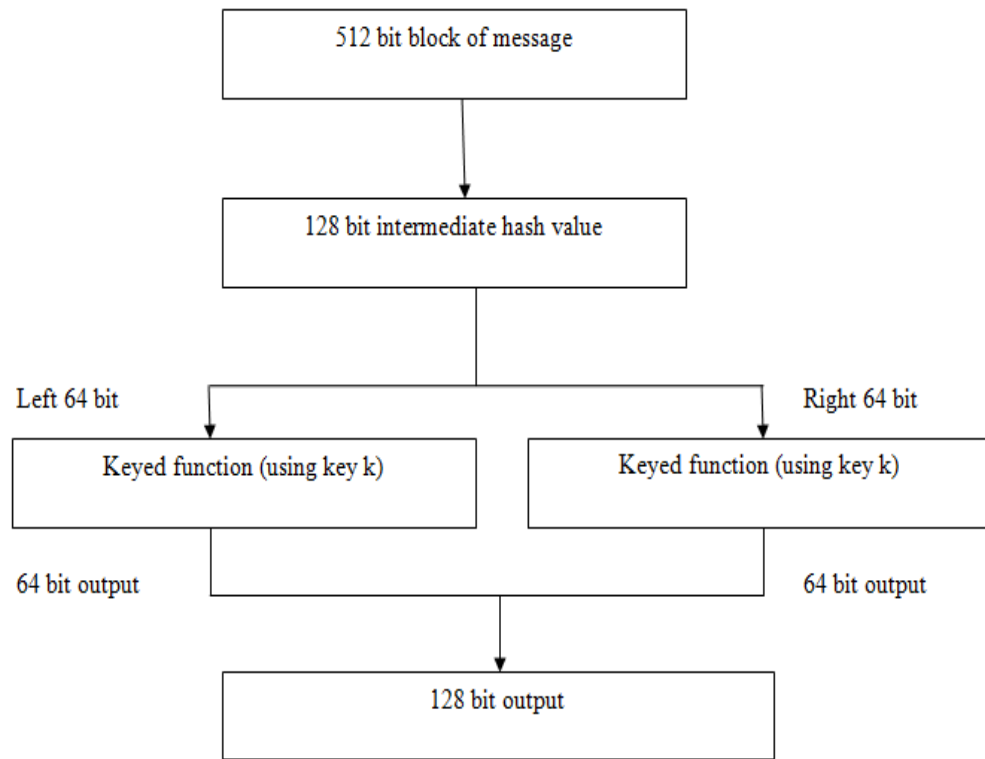
B<sub>2</sub> = Right 64 bits from output of hash value of 512 bit block

K= key used for each block

Designing and implementing a new secure hash function majorly includes fundamental two constructs- first, a compression function that may operate on any given input string in the form of block of data of a fixed length and then second to use the another function in cascade fashion so that output of compression function may extend the output length up to the string of arbitrary length. The design principal for the algorithm may be stated as: “make use of already proven techniques and build stronger one.” For this

- The proposed hash function is based around two established techniques- compression function and keyed function.
- The compression function accomplishes the requirement of providing basic building block for hash algorithm.
- To combine source authentication along with message integrity, keyed function is used.
- The proposed technique gives a solution for unauthenticated changes made in the input message and receiver does not receive it anymore on the assumption that it is coming from original sender only.

### 3.2 FLOW CHART OF THE PROPOSED HASH FUNCTION ALGORITHM



**Figure 6: Flow Chart of Proposed Algorithm**

### 3.3 PSEUDO CODE OF PROPOSED HASH ALGORITHM

Step 1: Start

Step 2: Put padding bits at the end of input message

Step 3: Put length of original message at the output of Step 2.

Step 4: Divide the output of Step 3 into L blocks of equal size. (512 bit blocks).

Step 5: Initialize 128 bit MD buffer

Step 6: Repeat Steps 7 to 11 for all 512 bit L blocks

Step 7: Calculate 128 bit hash value for the ith block

Step 8: Break output of Step 7 into two equal size blocks (64 bits each).

Step 9: Encrypt both blocks (outputs of Step7) using keyed block encryption function.

Step 10: Combine both 64 bit outputs calculated in Step 9.

Step 11: Use the output of Step 10 as CV for next 512 bit block.

Step 12: Transmit the final output of hashing of last block (Lth block) as final hash value to the receiver.

Step 13: End

### 3.4 DETAIL ANALYSIS OF THE PROPOSED HASH ALGORITHM

#### Step 1: Pad the Original Message-

The original message will be padded by specific number of bits, so that the length of input message after padding will become congruent to 448 modulo 512 ( $\text{length} \equiv 448 \pmod{512}$ ). For this purpose, first bit will always be 1 and remaining bits should always be 0. This is a compulsory step and thus, 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 will now be put at the end of the result of step 1. This length will be in 64 bit representation. After this step, the length of message will now be in multiples of 512.

#### Step 3: Initialize Buffer-

The algorithm uses a 128 bit buffer (four distinct words B1, B2, B3 and B4, 32 bit each), which will be initialized with following hexadecimal values:

B1 = 0 1 2 3 4 5 6 7

B2 = 8 9 A B C D E F

B3 = F E D C B A 9 8

B4 = 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 will also be stored in this.

#### Step 4: Initialize t-table-

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

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

#### Step 5: Four Secondary Functions-

$f_1(B_2, B_3, B_4) = (B_2 \wedge B_3) \vee (\neg B_2 \wedge B_4)$  for  $t = 0, \dots, 15$

$f_2(B_2, B_3, B_4) = (B_2 \wedge B_4) \vee (B_3 \vee \neg B_4)$  for  $t = 16, \dots, 31$

$f_3(B_2, B_3, B_4) = (B_2 \oplus B_3 \oplus B_4)$  for  $t = 32, \dots, 47$

$f_4(B_2, B_3, B_4) = B_3 \oplus (B_2 \vee \neg B_4)$  for  $t = 48, \dots, 63$

**Step 6: Order of words for processing:**

The processing will be done in 4 rounds. Each round will have 16 individual steps. For each step in each round, following sequence of words will be used for processing.

Round no. 1: ( $j_0, \dots, j_{15}$ ): Set  $j_0 = 0, j_1 = 1, j_2 = 2, j_3 = 3, j_4 = 4, j_5 = 5, j_6 = 6, j_7 = 7, j_8 = 8, j_9 = 9, j_{10} = 10, j_{11} = 11, j_{12} = 12, j_{13} = 13, j_{14} = 14, j_{15} = 15$

Round no. 2: ( $j_{16}, \dots, j_{31}$ ): Set  $j_{16} = 1, j_{17} = 6, j_{18} = 11, j_{19} = 0, j_{20} = 5, j_{21} = 10, j_{22} = 15, j_{23} = 4, j_{24} = 9, j_{25} = 14, j_{26} = 3, j_{27} = 8, j_{28} = 13, j_{29} = 2, j_{30} = 7, j_{31} = 12$

Round no. 3: ( $j_{32}, \dots, j_{47}$ ): Set  $j_{32} = 5, j_{33} = 8, j_{34} = 11, j_{35} = 14, j_{36} = 1, j_{37} = 4, j_{38} = 7, j_{39} = 10, j_{40} = 13, j_{41} = 0, j_{42} = 3, j_{43} = 6, j_{44} = 9, j_{45} = 12, j_{46} = 15, j_{47} = 2$

Round no. 4: ( $j_{48}, \dots, j_{63}$ ): Set  $j_{48} = 0, j_{49} = 7, j_{50} = 14, j_{51} = 5, j_{52} = 12, j_{53} = 3, j_{54} = 10, j_{55} = 1, j_{56} = 8, j_{57} = 15, j_{58} = 6, j_{59} = 13, j_{60} = 4, j_{61} = 11, j_{62} = 2, j_{63} = 9$

Processing will be done in blocks. Each block will be of 512 bit in length. A word is of 32 bits, thus, each block will be made up of 16 words ( $32 \times 16 = 512$ ).

**Step 7: Shifting-**

Shifting will be done in following amounts:

Round no. 1: Set  $S_0 = 7, S_1 = 12, S_2 = 17, S_3 = 22, S_4 = 7, S_5 = 12, S_6 = 17, S_7 = 22, S_8 = 7, S_9 = 12, S_{10} = 12, S_{11} = 22, S_{12} = 7, S_{13} = 12, S_{14} = 17, S_{15} = 22$

Round no. 2: Set  $S_{16} = 5, S_{17} = 9, S_{18} = 14, S_{19} = 20, S_{20} = 5, S_{21} = 9, S_{22} = 14, S_{23} = 20, S_{24} = 5, S_{25} = 9, S_{26} = 14, S_{27} = 20, S_{28} = 5, S_{29} = 9, S_{30} = 14, S_{31} = 20$

Round no. 3: Set  $S_{32} = 4, S_{33} = 11, S_{34} = 16, S_{35} = 23, S_{36} = 4, S_{37} = 11, S_{38} = 16, S_{39} = 23, S_{40} = 4, S_{41} = 11, S_{42} = 16, S_{43} = 23, S_{44} = 4, S_{45} = 11, S_{46} = 16, S_{47} = 23$

Round no. 4: Set  $S_{48} = 6, S_{49} = 10, S_{50} = 15, S_{51} = 21, S_{52} = 6, S_{53} = 10, S_{54} = 15, S_{55} = 21, S_{56} = 6, S_{57} = 10, S_{58} = 15, S_{59} = 21, S_{60} = 6, S_{61} = 10, S_{62} = 15, S_{63} = 21$

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

- For  $I = 0$  to  $n-1$  do (here,  $n$ = number of blocks)
- Divide  $M_i$  into words  $W_0, \dots, W_{15}$  where  $W_0$  is left most word.
- Initialization of 4 words  $B_1 B_2 B_3 B_4$ . Here each word is of 32 bit, i.e.

Total length =  $32 \times 4 = 128$  bit.

Assign  $B_1$  to  $B_1'$

Assign  $B_2$  to  $B_2'$

Assign B3 to B3'

Assign B4 to B4'

(' represents new value of buffer word)

d) For t = 0 to 63 do

Assign  $B2 + ((B1 + ft(B2, B3, B4) + Wjt + Kt) \lll St)$  to X

Assign B4 to B1

Assign B3 to B4

Assign B2 to B3

Assign X to B2

/\* end of loop in step d\*/

e) Increment of 4 words B1B2B3B4

Assign  $B1 + B1'$  to B1

Assign  $B2 + B2'$  to B2

Assign  $B3 + B3'$  to B3

Assign  $B4 + B4'$  to B4

f) Make two 64 bit blocks Y and Z from B1B2B3B4

Assign B1B2 to Y

Assign B3B4 to Z

g) Generate 64 bit key for internal keyed operation.

Out of these 64 bits, 8 will be used as parity bits and rest 56 bits are used as effective key. Out of this single 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 will 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.

Assign L to X

Assign R to L

Assign X to R

Now next L' and R' are produced as follows-

Assign R to L'

Assign  $L (+) f(R_{n-1}, K_n)$  to  $R'$

Here, (+) is addition modulo 232.

This process will be repeated for 16 times, each time with a different 48 bit key  $K$ .

Thus, Assign  $R_{n-1}$  to  $L_n$

Assign  $f(R_{n-1}, K_n) (+) L_{n-1}$  to  $R_n$

After sixteenth round of operation, again perform final permutation (swapping of left and right half), thus,

Assign  $L_n$  to  $X$

Assign  $R_n$  to  $L_n$

Assign  $X$  to  $R_n$

(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-**

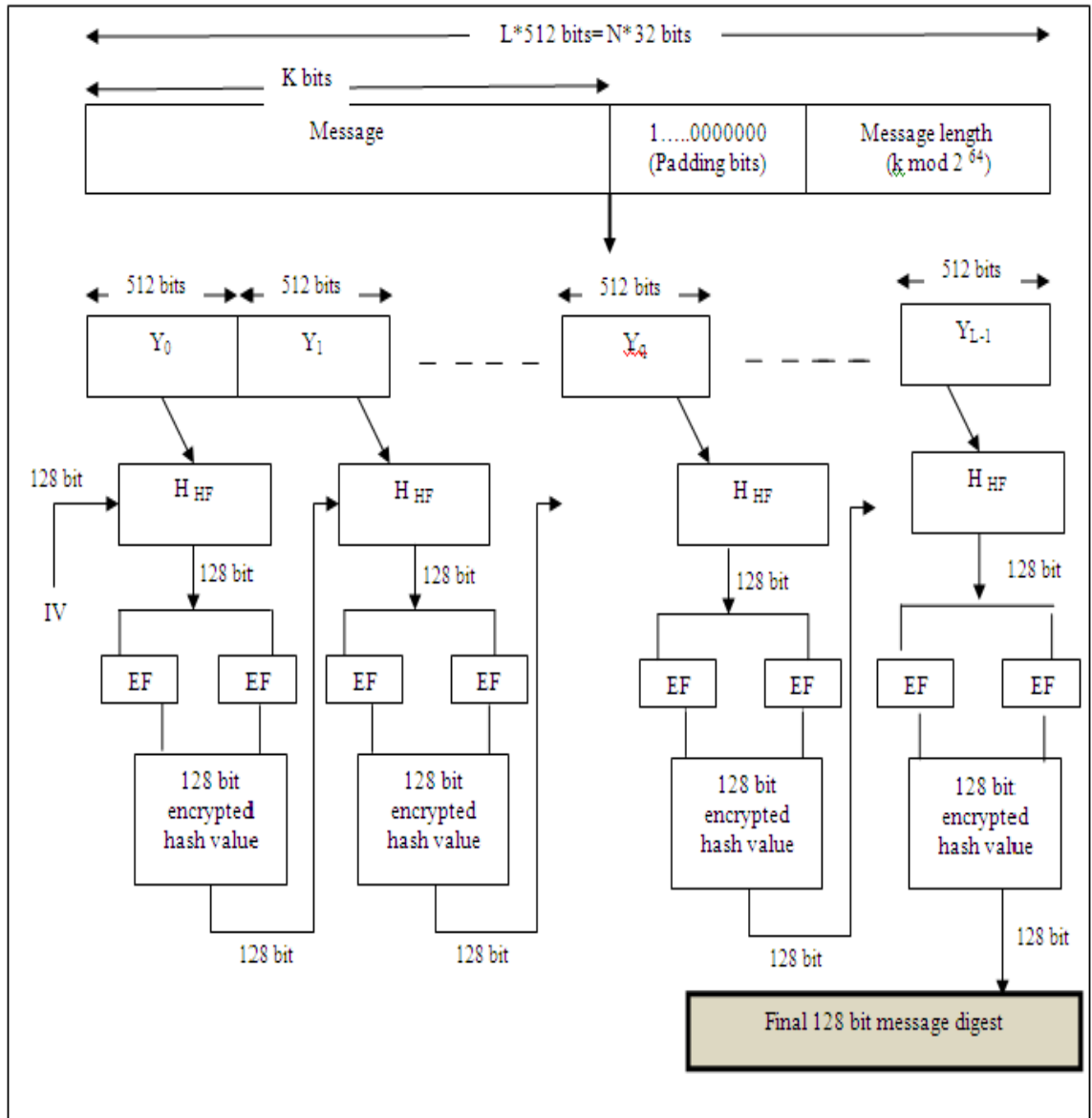
- a) Apply expansion permutation and output 48-bit data i.e assign  $E(R)$  to  $X$ .
- b) XOR 48 bit output with the round key i. e. assign  $X \wedge k$  to  $X'$ .
- c) Apply S boxes function on  $X'$  and generate output 32-bit data i. e. assign  $s(X')$  to  $X''$ .
- d) Apply the specific round permutation i. e. assign  $P(X'')$  to  $R'$ .

- i) Combine final 32 bit values of  $X$  and  $Y$ .

After combining two 64 bit blocks,  $Y$  and  $Z$  respectively, we will get one 128 bit block. These 128 bits will again stored in four 32 bit words  $B1B2B3B4$ .

/\* end of loop in step i. \*/

- j) After processing last 512 bit block, the final hash value is in  $B1B2B3B4$ , i.e. output is always 128 bit long digest.



**Figure 7: An illustrated view of processing of proposed hash function- having keyed function in between (HF= hash compression function and EF= keyed function)**

The Encryption Function EF works on a block of 64 bit message, that's why the 128-bit output of Hash Function HHF is subdivided into two parts, each of 64-bit. For the EF (Encryption Function), a key is used, with 64 bit length. Out of these 64 bits (eight 8-bit words) 8 bits are used as parity bits and remaining 56 bits are treated as actual effective key bits. The EF runs in 16 individual rounds with two permutation steps-one before first round and another after sixteenth round. For each round, a 48 bit key is used out of 56-bit effective key. Thus, 16 sub-keys are created from one 56-bit key. As

JavaScript supports only 32-bit integers, we need to represent all 48 or 64-bit integers by two 32-bit integers in an array. There are 72 quadrillion or more possible combinations of the keys. And for each message (or for each message block) a unique key may be selected.

To create 16 individual sub-keys, we begin with conversion of the 8 character string, which represents the key, into two integers. Bits of this 8 character string are reorganized according to permuted choice1 (Perm1).

The permutation choice gives the following bits as per their position-

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

**Figure 8: S-Box**

This shows that after permutation, the first bit of the key is generated from 57th bit of the original key, second bit of key is generated from 49th bit of original key bit sequence, and so on. A specific permutation sequence is being used for permutation in this process. The permutation involves an intelligent way of rotation and switching of bits between the given two integers. For example,  $\text{newint} = (\text{left} \ggg 4) \wedge \text{right}$ ;  $\text{right} = \text{right} \wedge \text{newint}$ ;  $\text{left} = \text{left} \wedge (\text{newint} \lll 4)$ ; it results in rotation of 4x4 blocks of bits.

Now, call the initial 28 bits of Perm1 as X, and remaining 28 bits as Y. These two parts X and Y are then left shifted according to a specific pattern until all sixteen 48 bit sub-keys are produced. For example, X1 and Y1 are produced by left shifting X and Y by



one bit place, X2 and Y2 are generated by left shifting X1 and Y1 by a further one place, and so on. These corresponding array and then the results are all added together.

### **3.5 EXPERIMENTAL SETUP**

For conducting this experiment, different format of data is taken for different cases as:

Case 1: For data below 50 KB

- Only text file is used

Case 2: For data in the range of 100 KB - 10 MB

- Pdf is used
- Jpg is used
- Multimedia (mp3, mp4, avi) is used

Programming Language used: JAVA

Specification of machine:

- Windows 7
- i5 Processor
- 2GB RAM
- 500 GB HDD

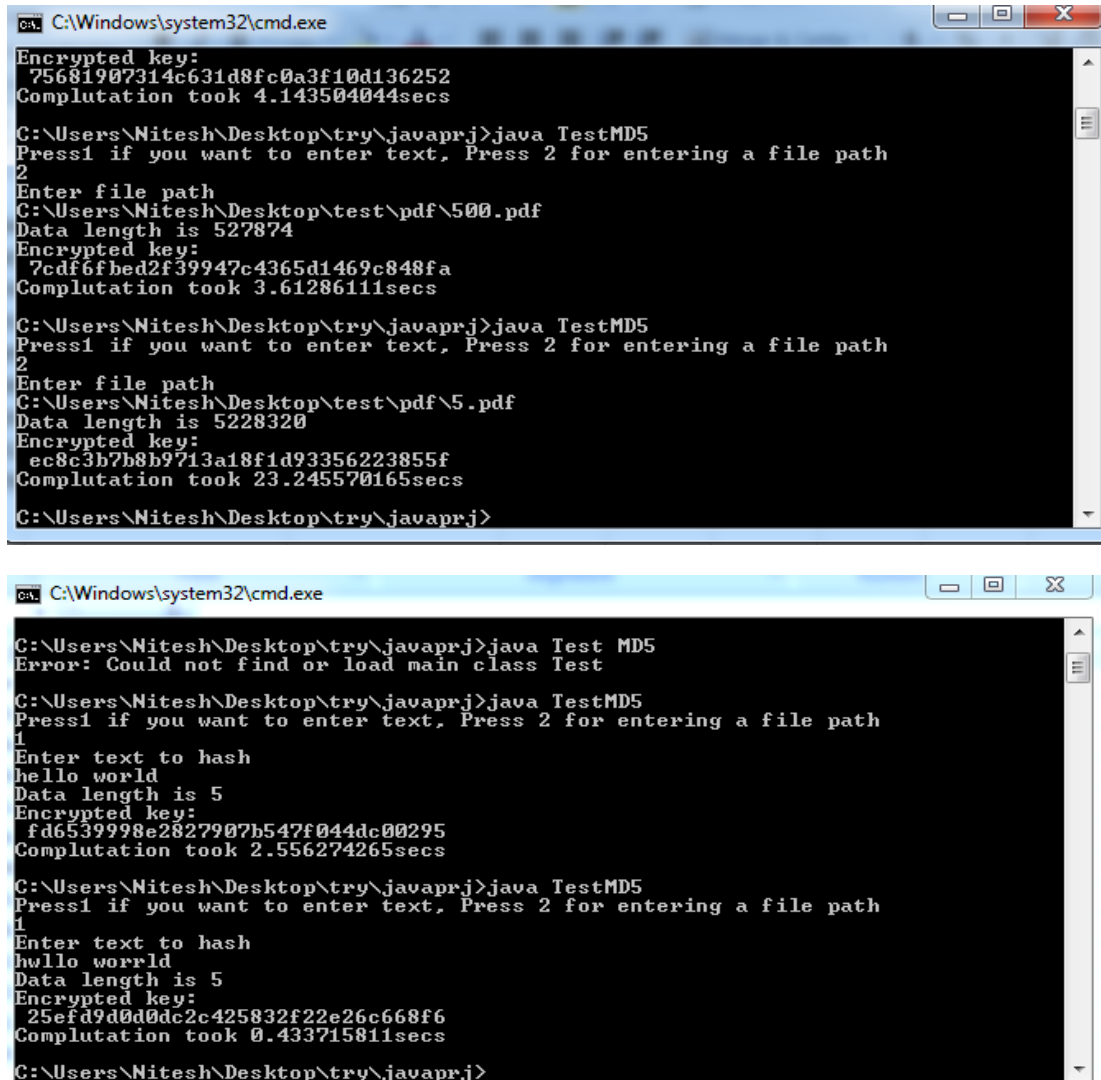
Program executed in Command Prompt.

## CHAPTER 4 RESULT AND ANALYSIS

### 4.1 RESULT

In order to become secure that the message is coming from the trusted sender and also to get security from various attack as first pre-image attack, second pre-image attack and brute force attack, time overhead should be considered. The proposed algorithm time complexity increases as compared with the keyless MD5 hash algorithm but the security level increase as described in the Table 6, 7, 9.

A large data set has been tested and run to get the result. The snapshot of the sample output is shown in the figure below:



```
C:\Windows\system32\cmd.exe
Encrypted key:
75681907314c631d8fc0a3f10d136252
Computation took 4.143504044secs

C:\Users\Nitesh\Desktop\try\javaprj>java TestMD5
Press1 if you want to enter text, Press 2 for entering a file path
2
Enter file path
C:\Users\Nitesh\Desktop\test\pdf\500.pdf
Data length is 527874
Encrypted key:
7cdf6fbed2f39947c4365d1469c848fa
Computation took 3.61286111secs

C:\Users\Nitesh\Desktop\try\javaprj>java TestMD5
Press1 if you want to enter text, Press 2 for entering a file path
2
Enter file path
C:\Users\Nitesh\Desktop\test\pdf\5.pdf
Data length is 5228320
Encrypted key:
ec8c3b7b8b97713a18f1d93356223855f
Computation took 23.245570165secs

C:\Users\Nitesh\Desktop\try\javaprj>

C:\Windows\system32\cmd.exe
C:\Users\Nitesh\Desktop\try\javaprj>java Test MD5
Error: Could not find or load main class Test
C:\Users\Nitesh\Desktop\try\javaprj>java TestMD5
Press1 if you want to enter text, Press 2 for entering a file path
1
Enter text to hash
hello world
Data length is 5
Encrypted key:
fd6539998e2827907b547f044dc00295
Computation took 2.556274265secs

C:\Users\Nitesh\Desktop\try\javaprj>java TestMD5
Press1 if you want to enter text, Press 2 for entering a file path
1
Enter text to hash
hwlllo world
Data length is 5
Encrypted key:
25efd9d0d0dc2c425832f22e26c668f6
Computation took 0.433715811secs

C:\Users\Nitesh\Desktop\try\javaprj>
```

Figure 9: Screenshots of the program output

The Figure 7 shows that this program is run by the command line and data can be entered in both the way by direct entering or by giving the path of the file. For the size less than 50 KB, only text type of data is used as it becomes difficult to get the less than 50 KB especially for multimedia (mp3, mp4, avi) type of data.

#### **4.2 SECURITY LEVEL ACHIEVED BY THE PROPOSED HASH ALGORITHM**

A strong cryptographic hash function  $H$  is usually expected to satisfy a number of requirements, namely collision resistance, preimage resistance, second preimage resistance. There are three important attacks on hashes [7]:

- A "collision attack" allows an attacker to find two messages  $M_1$  and  $M_2$  that have the same hash value in fewer than  $2^{(L/2)}$  attempts [7].
- A "first-preimage attack" allows an attacker who knows a desired hash value to find a message that results in that value in fewer than  $2^L$  attempts [7].
- A "second-preimage attack" allows an attacker who has a desired message  $M_1$  to find another message  $M_2$  that has the same hash value in fewer than  $2^L$  attempts [7].

All individual components of the proposed algorithm possess their respective security criteria, which ensure that the algorithm is secure and collision free. Few arguments can be given for security of the algorithm:-

- Mathematically secondary functions  $f_1$ ,  $f_2$ ,  $f_3$  and  $f_4$  are non-invertible and non-linear [Section 3.4 step 5].
- If individual bits of  $B_2$ ,  $B_3$  and  $B_4$  are independent of each other, this implies that the overall bits of  $f(B_2, B_3, B_4)$  are also independent of each other. It ensures the desired one-way property of hash function.
- During processing, in each of the round, accessing sequence for input words is different.
- In padding step, the padding is always done by 1 and then the number of zeros in the output of preprocessing phase to restrict the fixed point attacks in which the attacker's main focus is to generate second pre-image or collisions by insertion of extra blocks into the input.

- In each step input is taken from the output of previous step. Thus, changes at any one place in any one of the blocks; it will definitely affect the final output of algorithm. Thus, two different messages will never result in same output. It proves one more desirable hash property- the second pre-image resistance.
- Number of computations to perform Collisions and attacks depend on key size and digest size both. Thus for 128 bit hash Proposed Hash  $2^{192}$  computations are required to perform brute force attack.
- Algorithm works on basic functions, such as modular arithmetic, XOR, addition, left shift, right shift, simple permutation etc. Thus, it does not require increased time requirement for processing.
- The required t-table and all 48 bit 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 speeding up the execution of the algorithm.
- Use of XOR makes sure that output depends on all bits, rather than on neighboring ones only.

### **Key Generation and Usage**

All popular existing keyed hash functions use either of two settings- Dedicated-Key setting or 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, 16 different key combinations are used 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 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, 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.

### 4.3 COMPARISON WITH EXISTING APPROACHES

**Table 9: Comparison of the Proposed algorithm with the various enhanced approached algorithm on the basis of different parameters.**

Hash Algorithm	Block size(bit)	Output Size (bits)	Round	Brute force Attack Operation	Factors for enhancing security
MD5 [23]	512	128	64	$2^{64}$	Round
SHA-1 [23]	512	160	80	$2^{64}$	Increases in Output size and Increase in Round
An Enhanced Message Digest Hash Algorithm for Information Security [24]	512	160	160	$2^{160}$	Increases in Output size and Increase in Round
One Improved Hash Algorithm Based on MD5 and SHA1 [10]	512	512	64	$2^{128}$	Increases in Output size
Proposed	512	128	128	$2^{192}$	Increase in Round Use of Key

When comparing with some of the existing approaches, it is found that most of the existing approaches does not uses key to make MD5 secure. So still they does not provides source authentication whereas the proposed algorithm uses a key of 128 bit which provides both the message integrity and source authentication and according to Kasgar and dhariwal [25], the algorithm is found to be secure enough if it uses a key of 128 bit. So it can be proved that the proposed hash algorithm is better in security approach than the traditional keyless hash algorithm.

## PROPERTIES OF PROPOSED HASH FUNCTION WITH RESPECT TO EXISTING HASH FUNCTION

**Table 10: Properties of existing hash function and Proposed hash function**

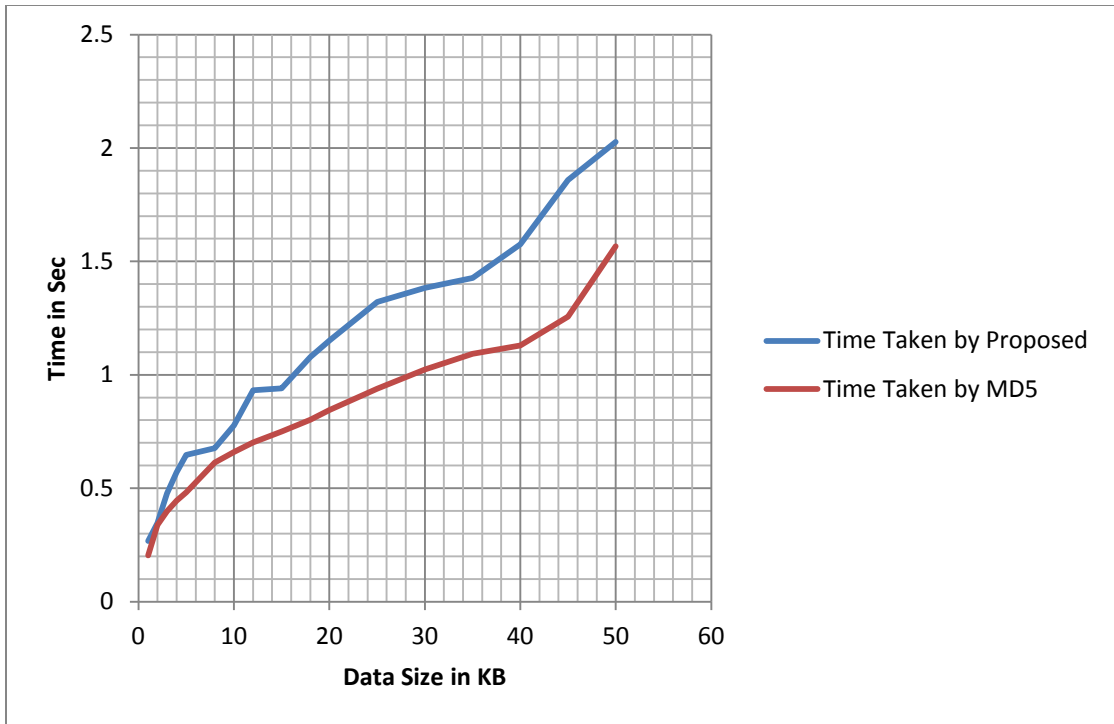
Properties of existing hash function	Properties of Proposed hash function
MD2, MD4, MD5, SHA family, RIPEMD, BLAKE, TAV etc message detection codes do not use “key”. Only message and initialization values are used as input to the function.	It uses “key” along with message and initialization values are used as input to the function.
MD2, MD4, MD5, SHA family, etc message detection codes provide only message integrity.	It provides message integrity and source authentication both.
Message Authentication codes such as HMAC and nested MAC use key but the simply prefix or postfix the key to the message, So it is easy to forge.	Key is used in each round of operation, not just pre or post fixed, which makes it difficult to forge.
In message authentication codes such as HMAC and nested MAC, key can easily be traced easily if message length is known in advance.	Key cannot be traced from final digest as it becomes integral part of hash function.
Collisions can be performed as only message and IV is put to the function.	To find collision is more difficult because of key
Number of computations to perform Collisions and attacks depend on digest size only. Thus for 128 bit MD5 $2^{128}$ computations are required to perform brute force pre-image and brute force second pre-image attack.  For RIPEMD-160, SHA-1, SHA-0, required calculations are minimum $2^{160}$ .	Number of computations to perform Collisions and attacks depend on key size and digest size both. Thus for 128 bit Proposed Hash $2^{192}$ computations are required to perform brute force attack.

#### 4.4 ANALYSIS OF TIME REQUIREMENT

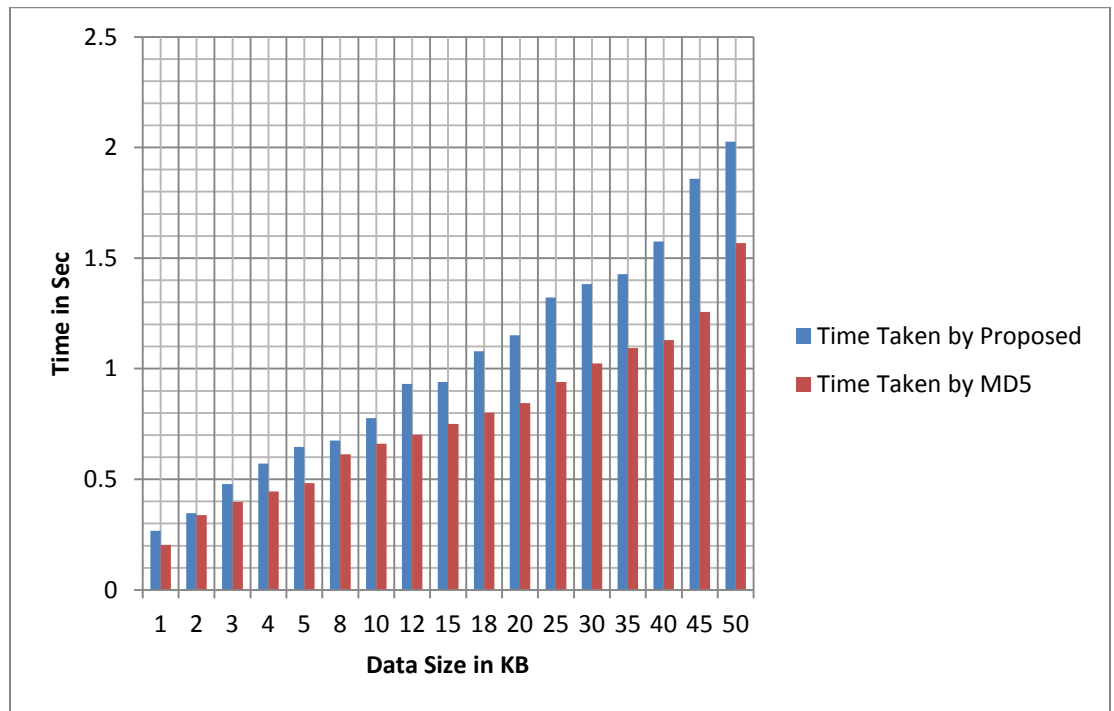
For the data less than 50 KB, only text type of data is used for experimentation. Every data is tested three times and an average is taken. The output along with the line graph and bar graph for the data is summarized as:

**Table 11: Time taken by MD5 hash and Proposed algorithm in the range (1KB-50KB) data.**

Data Size in KB	Time Taken by Proposed Algorithm in Second	Time Taken by MD5 in Second
1	0.467037956	0.203622368
2	0.346586704	0.338110466
3	0.477840691	0.398090032
4	0.571099918	0.445129448
5	0.646240773	0.482157356
8	0.675861606	0.613133924
10	0.776960662	0.660249677
12	0.931250181	0.701610605
15	0.939926339	0.750443612
18	1.078135002	0.80159429
20	1.151103277	0.844284213
25	1.321655376	0.939422
30	1.382957886	1.024267855
35	1.427354081	1.09326926
40	1.575217228	1.129504505
45	1.858272684	1.256137605
50	2.026442377	1.56723058



**Figure 10: Line Diagram of Time taken by MD5 hash and Proposed algorithm in the range (1KB- 50KB) data.**



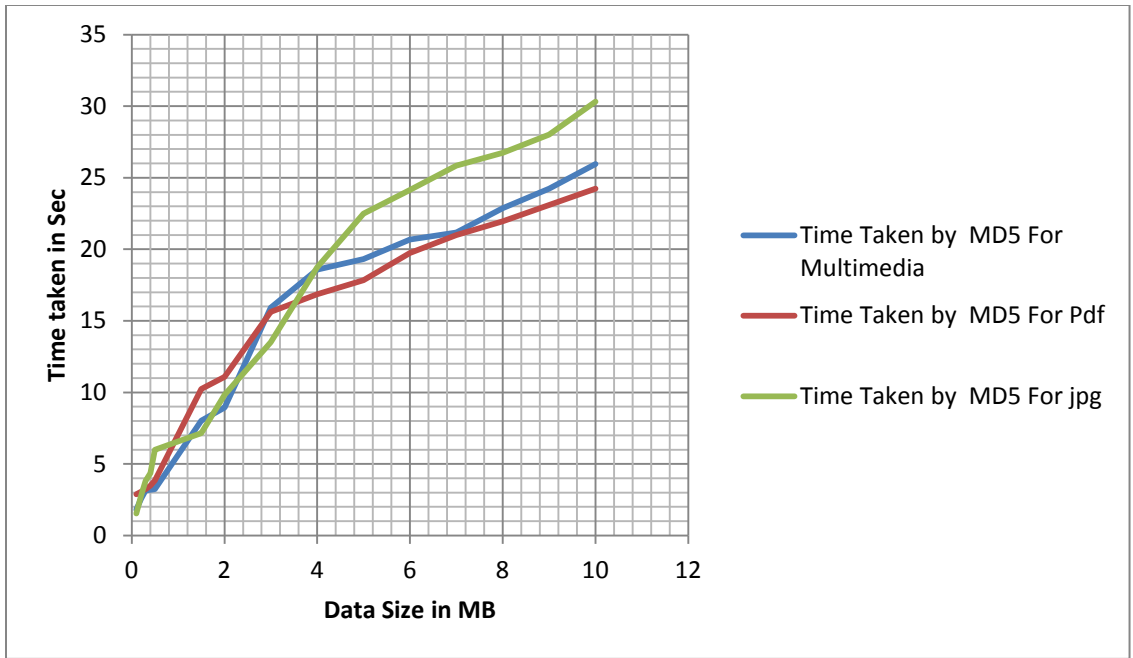
**Figure 11: Bar Diagram of Time taken by MD5 hash and Proposed algorithm in the range (1KB- 50KB) data.**



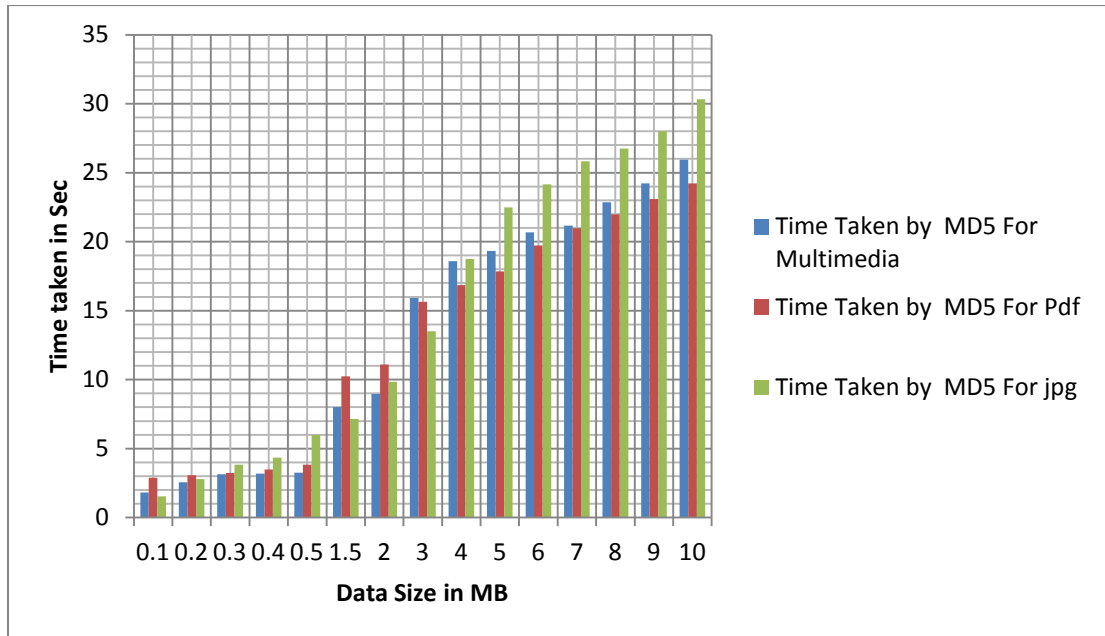
For the data greater than 100 KB experiment is done on different format like Multimedia (mp3, mp4), pdf and jpg types of data since it is very difficult to find the multimedia format of data in less than 100 KB size. Every data is tested three times and average is taken. The tested result is shown below.

**Table 12: Time Taken by Different format (Multimedia, Pdf, jpg) of data by MD5 Hash Algorithm**

Data size in MB	Time Taken by Different format of data by MD5 in Second		
	Multimedia (mp3, mp4, avi)	Pdf	jpg
0.1	1.8127	2.8903	1.5342
0.2	2.5613	3.0653	2.8022
0.3	3.1416	3.2412	3.8285
0.4	3.1893	3.4916	4.3482
0.5	3.2512	3.8393	5.9856
1.5	8.0213	10.2342	7.1602
2	8.952	11.0935	9.8467
3	15.9154	15.6352	13.4971
4	18.5782	16.8469	18.7456
5	19.3216	17.8476	22.4971
6	20.6739	19.7366	24.1523
7	21.1734	20.9832	25.8367
8	22.8646	21.9654	26.746
9	24.2341	23.0913	28.0173
10	25.9456	24.2315	30.3215



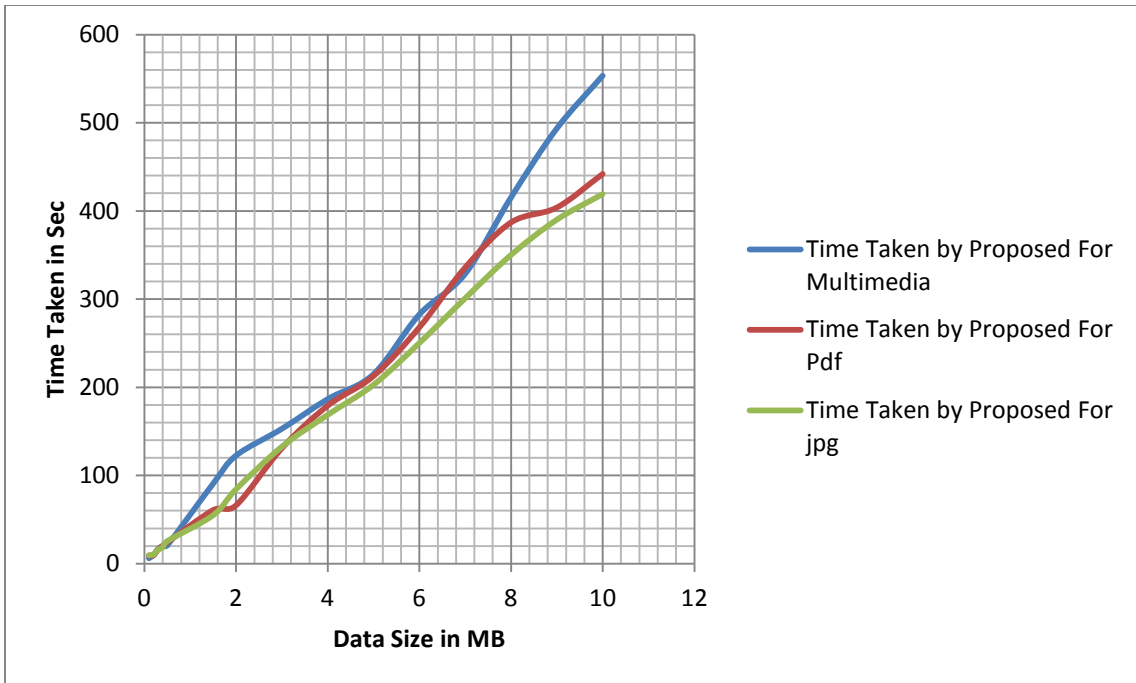
**Figure 12: Line Diagram Time Taken by Different format (Multimedia, Pdf, jpg) of data by MD5 Hash Algorithm**



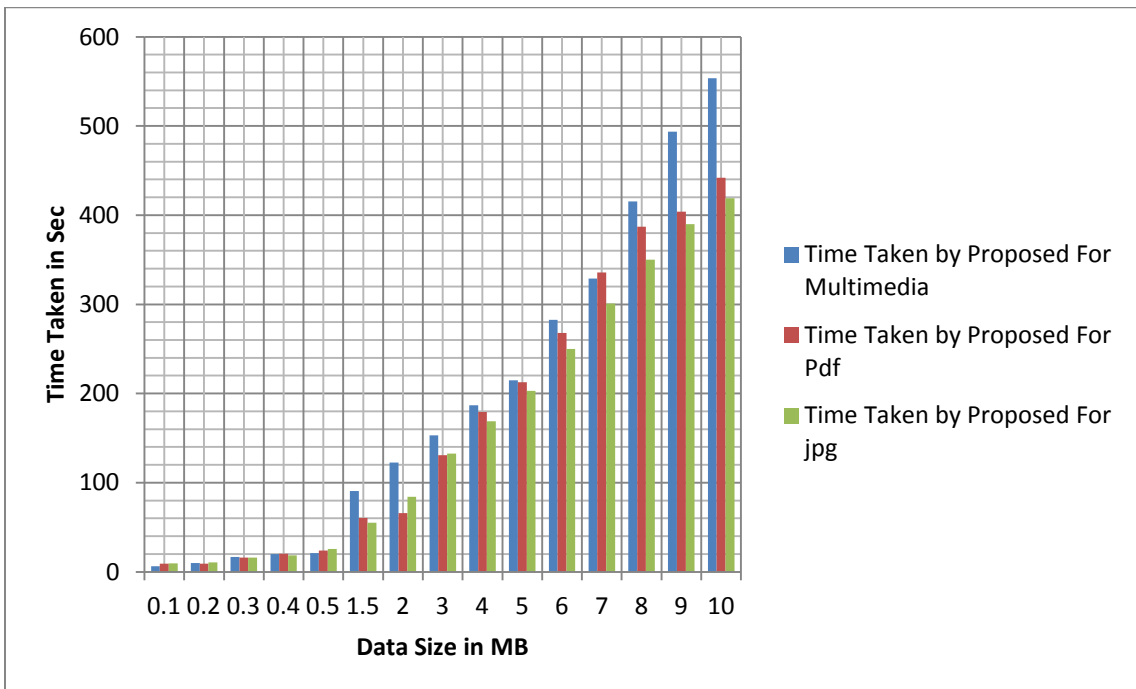
**Figure 13: Bar Diagram Time Taken by Different format (Multimedia, Pdf, jpg) of data by MD5 Hash Algorithm**

**Table 13: Time Taken by Different format (Multimedia, Pdf, jpg) of data by Proposed Algorithm**

Data size in MB	Time Taken by Different format of data by Proposed in Second		
	Multimedia	Pdf	jpg
0.1	6.2172	9.0321	9.6734
0.2	9.8355	9.3254	10.4714
0.3	16.7466	15.8734	15.8241
0.4	20.0191	20.123	18.5106
0.5	20.8685	24.027	25.6849
1.5	90.7372	60.6534	54.9376
2	122.6345	65.9876	84.3073
3	153.0185	130.7635	132.5996
4	186.7352	179.0764	168.6971
5	214.9371	212.7672	202.7672
6	282.6923	267.9473	250.1125
7	328.8362	335.8462	300.8741
8	415.3425	386.9473	349.9472
9	493.6352	403.9364	389.9286
10	553.4534	442.0205	418.9268



**Figure 14: Line Diagram Time Taken by Different format (Multimedia, Pdf, jpg) of data by Proposed Hash Algorithm**



**Figure 15: Bar Diagram Time Taken by Different format (Multimedia, Pdf, jpg) of data by Proposed Hash Algorithm**

As shown in Figure 13, 14 there is some time difference between various format of data for both MD5 hash and Proposed hash. One of the main reason behind this difference is that the data length taken here is not exactly same, but an approximate size is taken for the all format of data (eg,1.1MB and 0.98 MB size is taken as 1 MB).

## **ANALYSIS**

Figure 6, 7 clearly shows that the time taken by MD5 hash is not much greater than that of Proposed algorithm in the range (1KB- 50KB) data. The time taken by proposed algorithm and MD5 hash is linearly proportional to the size of data. The time taken by proposed algorithm is only 15-20% greater than that of MD5 hash algorithm.

Figure 8, 9 shows that the time taken by proposed algorithm increases upto 400% as the data size increases to 500 KB. The main reason behind increasing the time is that the algorithm runs on fixed size block of 512 bit, from which 128 bit intermediate hash is generated which is further divided into two equal half blocks and then encryption function is applied on each 64 bit block which is then combined to get 128 bit hash. So as the data size increases, the number of 512 bit blocks also increases for the further processing of data, but in case of MD5 hash algorithm there is no use of any key for the encryption function. That's why the time required to get the output hash is less than that of the proposed hash algorithm.

For the data size above 100 KB, various formats of data as .jpg, .pdf, multimedia (.mp3, .mp4, .avi) is tested for both the algorithm (MD5, proposed).The result can be seen from the Figure 10, 11 and Figure 12, 13. Here in both the cases there is some time difference in between them for different formats. The main reason behind this difference is that the data size is not same for all format of data, as the data set is created manually, it becomes very difficult to get data of exact size especially for the multimedia (mp3, mp4) type of data.

## 4.5 COMPUTATIONAL COMPLEXITY

### Initialize the buffer

$B1 = 0\ 1\ 2\ 3\ 4\ 5\ 6\ 7$   
 $B2 = 8\ 9\ A\ B\ C\ D\ E\ F$   
 $B3 = F\ E\ D\ C\ B\ A\ 9\ 8$   
 $B4 = 7\ 6\ 5\ 4\ 3\ 2\ 1\ 0$

**O(1)**

For(i=0 to 63)

// initialize the T table

$k_t = 2^{32} \sin(t+1)$

**O(1)**

### Four Secondary Functions-

For(t=0 to 15)

$f1(B2, B3, B4) = (B2 \wedge B3) \vee (\neg B2 \wedge B4)$

**16\*c1**

For(t=16 to 31)

$f2(B2, B3, B4) = (B2 \wedge B4) \vee (B3 \vee \neg B4)$

**16\*c2**

For(t=32 to 47)

$f3(B2, B3, B4) = (B2 \oplus B3 \oplus B4)$

**16\*c3**

For(t=48 to 63)

$f4(B2, B3, B4) = B3 \oplus (B2 \vee \neg B4)$

**16\*c4**

### Order of words for processing:

Round1(Algo1)  
 Round1(Algo2)  
 Round1(Algo3)  
 Round1(Algo4)

**16\*c1**  
**16\*c2**  
**16\*c3**  
**16\*c4**

### Shifting-

Round 1  
 Round 2  
 Round 3  
 Round 4

**O(1)**

**Processing of message in sixteen 32-bit word (512 bit) blocks-**

For ( i= 0 to n-1) do

Divide  $M_i$  into words  $W_0, \dots, W_{15}$  where  $W_0$  is left most word.

Initialization of 4 words  $B_1 B_2 B_3 B_4$ . Here each word is of 32 bit, i.e,

$B_1 \rightarrow B_1'$

$B_2 \rightarrow B_2'$

$B_3 \rightarrow B_3'$

$B_4 \rightarrow B_4'$

$O(1)$

$16n+c$

For (t=0 to 63)

Assign  $B_2 + ((B_1 + f_t(B_2, B_3, B_4) + W_{jt} + K_t) \lll S_t) \rightarrow X$

Assign  $B_4$  to  $B_1$

Assign  $B_3$  to  $B_4$

Assign  $B_2$  to  $B_3$

Assign  $X$  to  $B_2$

$64*c$

End for loop

**Increment of 4 words  $B_1 B_2 B_3 B_4$**

$B_1 + B_1' \rightarrow B_1$

$B_2 + B_2' \rightarrow B_2$

$B_3 + B_3' \rightarrow B_3$

$B_4 + B_4' \rightarrow B_4$

$O(1)$

**Make two 64 bit blocks  $Y$  and  $Z$  from  $B_1 B_2 B_3 B_4$**

Assign  $B_1 B_2 \rightarrow Y$

Assign  $B_3 B_4 \rightarrow Z$

$O(1)$

**Operations on  $Y$  and  $Z$  blocks**

Divide  $Y$  into  $L_Y$  and  $R_Y$

Divide  $Z$  into  $L_Z$  and  $R_Z$

$L \rightarrow X$

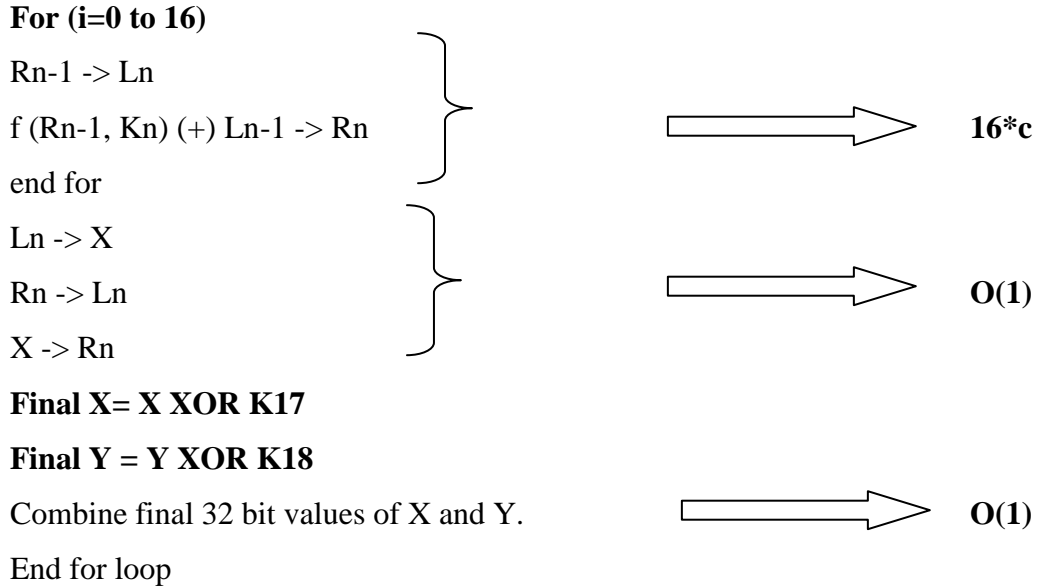
$R \rightarrow L$

$X \rightarrow R$

$R \rightarrow L'$

$L (+) f(R_{n-1}, K_n) \rightarrow R'$

$O(1)$



$$\begin{aligned}
 \text{Total Time complexity} &= O(1) + O(1) + (16*c1 + 16*c2 + 16*c3 + 16*c4) + (16*c1 + \\
 &16*c2 + 16*c3 + 16*c4) + O(1) + 16n (O(1) + 64*c + O(1) + O(1) + O(1) + O(1) + 16*c \\
 &+ O(1) + O(1)) \\
 &= O(n)
 \end{aligned}$$

The time complexity of the proposed hash algorithm is linear i.e. the time taken to encrypt data is directly proportional to the size of data. One of the reason that the proposed algorithm takes more time than the existing MD5 hash function is that

- MD5 hash is keyless while in the proposed solution, 16 different key combinations are used in an iteration of compression function, individually on two word combinations Y and Z respectively which obviously is more time consuming than keyless hash algorithm, 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 [see section 3.4 step 8(h), (i)].
- As the data size increases the number of blocks of 512 bit also increases upon which the encryption function has to applied which therefore increases time taken to encrypt the data.



## 4.6 DISCUSSION

MD5 is a well-known and widely-used cryptographic hash function. It has received renewed attention from researchers subsequent to the recent announcement of collisions found by Wang et al [14]. MD5 hash function is fast and one-way hash function, and provides security in case adversary modifies data in unauthenticated manner. But it has been observed that only message integrity does not guarantee sufficient security against all of the proven attacks. For example, if the algorithm for generating the code is known, an adversary can generate the correct code after modifying the data, thus, ordinary error detecting codes are not adequate. Intentional modification is undetectable with such codes. That is, suppose a message  $X$  is sent by sender along with its calculated hash value  $h$ . After interruption, intruder fetches the message, and changes this  $X$  into  $X'$ . At the same time, he may also get copy of  $h$  and recalculates new hash  $h'$  for new message  $X'$ , and then transmits it to the receiver. At the receiving end, the receiver recalculates hashes on received version of message i.e.  $X'$ . Now, it will result in verified one, which is not true [23].

However, simple symmetric encryption techniques can be used to produce a cryptographic checksum. This checksum can be used to protect against unauthorized data modification which may be either accidental or intentional. Additionally, if a hashing scheme is combined with a block cipher encryption algorithm, it can be made more secured and stronger against attacks. The hash function  $h$  is defined in such a manner that from the message  $X$ ,  $h(X)$  can be calculated easily, but if one knows  $h(X)$ , than it should be infeasible to find even one message  $X$  that will generate this value. Moreover, calculating any other message  $M'$  that produces the same hash value, i.e.,  $h(X) = h(X')$ , must also be infeasible. The hash value may then be given to any strong block encryption function, whose key is already known to sender and receiver both. The sender will use this key for calculating hash at its end. And the same corresponding key will be used by the receiver to revert the transformation and restore the value  $h(X)$ . At the receiving end, the function  $h$  is applied to the received message  $X$ , and then he compares two values of  $h(X)$ . Only if the message is original and not modified after generating hash by sender, these two hash values will appear to be equal [23].

The proposed hash algorithm uses a symmetric key of 128 bit which provides both the message integrity and source authentication. It also prevents intentional modification of data.

In the proposed hash algorithm, in each step, input is taken from the output of previous step. Thus, if changes at any one place in any one of the blocks, it will definitely affect the final output of algorithm. Thus, two different messages will never result in same output. It proves one more desirable hash property- the second pre-image resistance.

This solution use two basic rules: padding and fixed initialization vector. Thus, it is also safe from fixed point attack and second pre-image collision attack. At the same time, there exists no method for getting original message from hash value. It also uses the concept of key for generating hash value, which implies that 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, and it is delivered to them by trusted Key Distribution Center (KDC).

To perform brute force attack on an n-bit message digest an attacker must take about  $2^n$  computations of hash function to obtain the pre-image with a significant percent of probability. Thus for proposed hash (a 128 bit digest) it takes minimum  $2^{128}$  computations for an attacker before performing a successful brute force pre-image attack and brute force second pre-image attack. But as proposed hash also uses a 128 bit key, its required minimum calculations are  $2^{192}$ , which is better than MD-5. Similarly, it also takes  $2^{192}$  computations for brute force collision attack which in practical is not feasible to actually produce this huge amount of computational power today.

## **CHAPTER 5**

### **CONCLUSIONS AND FUTURE WORK**

#### **5.1 CONCLUSIONS**

From the above discussion it can be concluded that, even if the time taken by the proposed algorithm is 15-20% more than of the existing keyless MD5 hash algorithm for data range between 50 KB-100 KB, the security level increases from  $2^{64}$  to  $2^{192}$  operations to perform brute force attack. The proposed algorithm can be used instead of MD5 hash algorithm because:-

1. The proposed hash algorithm provides both message integrity and source authentication as compared with the existing keyless MD5 hash which provides message integrity only but not the source authentication.
2. The proposed hash algorithm is very much resistant to pre-image attack attacks in which the attacker's main focus is to generate second pre-image or collisions by insertion of extra blocks into the input.
3. The proposed hash algorithm is resistant to second pre-image attack so that two different messages will never result in same output.
4. It requires more number of operation to perform brute force attack as the traditional keyless MD5 hash algorithm needs only  $2^{64}$  operation to perform collision or second preimage attack, proposed hash algorithm requires  $2^{192}$  operations to perform collision or second preimage attack.
5. Even it takes 15-20% of time more than that of existing keyless MD5 hash for low size data.

Thus, it can be concluded that the proposed hash algorithm is enhanced than the existing hash and can be used in place of MD5 hash algorithm. As far as time is concerned it is best suited for the small size below 100 KB as Password hashing, Virus Checking, data of IOT, sensors data, Message Authentication Code, small size military information etc. however in some cases where the security of a document is main priority irrespective of time it is suggested to use the proposed hash algorithm in spite of MD5 hash.

## **5.2 LIMITATIONS AND FUTURE WORK**

During the performance analysis of the proposed algorithm, it is found that the time taken to encrypt the data size exceeding (100 KB-150 KB) increases by more 30% - 50% and reaches upto 400% as the data size reaches more than 10 MB and above. This can be overcome if we may use more computational powerful machine.

The algorithm uses keyed function for a block of 64 bits. In future we may also perform the keyed function for smaller or larger block size and see whether it improves the security or time taken for overall processing of message. This algorithm uses key of 128 bits. In future it can be tested for 256 bits and more to check the level of security and the time taken to encrypt the data.

From the point of view of designing, there is always requirement of new types of hash functions. These new functions should offer a high level of security as well as their performance should also be better than previous ones. To possess more provable security properties for hash functions would also be desirable always. For block cipher based hash functions it is still an open problem to produce a secure hash

## REFERENCES

- [1] P. kumar and V. Agarwal, "Crypt Analyzing of Message Digest Algorithms MD5 Using Quadratic Salt," *International Journal Of Engineering And Computer Science ISSN*, vol. 2, no. 1, pp. 36-42, August,2014.
- [2] L. Pathak, B. K. Sharma and R. Sharma, "Breaking of Simplified Data Encryption Standard," *Global journal of computer science and technology*, vol. 12, no. 5, p. 7, March 2012.
- [3] Somboonpattanakit, S. Boonkrong and Chaowalit, "Dynamic Salt Generation and Placement for," *IAENG International Journal of Computer Science*, vol. 43, no. 1, p. 10, 29th Februrary 2016.
- [4] N. Ogini, N. Oluwole and O. Ogwara, "Securing Database passwords using a combination of hashing and salting techniques," *IPASJ International Journal of Computer Science (IJCS)*, vol. 2, no. 8, pp. 2-6, August,2014.
- [5] S. Mohammed, "Modified Key Model of Data Encryption Standard," University of Anbar, Iraq, 2014.
- [6] S. Singh, S. K. Maakar and D. Kumar, "Enhancing the Security of DES Algorithm Using Transposition Cryptography Techniques," *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 3, no. 6, pp. 1-8, June, 2013.
- [7] R. Roshdy, M. Fouad and M. Aboul-Dahab, "Design and Implementation a New Security Hash Algorithm Based On MD5 and SHA-256," *International Journal of Engineering Sciences & Emerging Technologies*, vol. 6, no. 1, pp. 29-36, August, 2013..
- [8] V. Thapar and P. Walia, "Implementation of New Modified MD5-512 bit Algorithm for Cryptography," *International Journal of Innovative Research in Advanced Engineering (IJIRAE)*, vol. 1, no. 6, pp. 87-96, July,2014.
- [9] D. Sharma and P. Sarao, "Implementation of Md5- 640 Bits Algorithm," *International Journal of Advance Research in Computer Science and Management Studies*, vol. 3, no. 5, pp. 286-294, May,2015.

- [10] X. Chan and G. Liu, "Discussion of One Improved Hash Algorithm Based on MD5 and SHA1," *IOSR Journal of Computer Engineering (IOSRJCE)*, vol. 2, no. 1, pp. 36-42, October, 2007.
- [11] K. Singh and C. Goel, "Using MD5 AND RSA Algorithm Improve Security in MANETs Systems," *International Journal of Advances in Science and Technology (IJAST)*, vol. 2, no. 2, pp. 48-53, June 2014.
- [12] Dr. P. Mahajan and A. Sachdeva, "A Study of Encryption Algorithms AES, DES and RSA for Security," *Global Journal of Computer Science and Technology Network, Web & Security*, vol. 13, no. 15, pp. 2-9, 2013.
- [13] T. Xie and F. Liu, "Fast Collision Attack on MD5," The Center for Soft-Computing and Cryptology, China, 2014.
- [14] R. Jasek, L. Sarga and R. Benda, "Security Review of the SHA-1 and MD5 Cryptographic Hash Algorithms," ISBN, Zlin, August, 2013.
- [15] Hamdan.O.Alanazi, B.B.Zaidan, A.A.Zaidan, M. Hamid A.Jalab and Y. Al-Nabhani, "New Comparative Study Between DES, 3DES and AES within Nine Factors," *Journal Of Computing*, vol. 2, no. 3, pp. 2-12, March, 2010.
- [16] M. Agrawal and P. Mishra, "A Comparative Survey on Symmetric Key Encryption Techniques," KTH CSC, Sweden, 2008.
- [17] Nadeem, Aamer and D. M. Y. Javed, "A Performance Comparison of Data Encryption Algorithms," Pakistan, April, 2015.
- [18] Elminaam, D. S. Abdul, H. M. A. Kader and M. M. Hadhoud, "Performance Evaluation of Symmetric Encryption Algorithms," *IJCSNS International Journal of Computer Sci 280 ence and Network Security*, vol. 8, no. 12, pp. 281-288, December, 2008.
- [19] S. M. Seth and R. Mishra, "Comparative Analysis Of Encryption Algorithms For Data Communication," *International journal of Computer Science and Technology*, vol. 2, no. 2, pp. 293-294, june, 2011.
- [20] P. Kuppuswamy, P. M. Appa and D. S. Q. Y. Al-Khalidi, "A New Efficient Digital Signature Scheme Algorithm based on Block Cipher," *IOSR Journal of Computer*

- Engineering (IOSRJCE)*, vol. 7, no. 1, pp. 47-52, December, 2012.
- [21] S. R. Masadeh, S. Aljawarneh, Turab, Nedal, Abuerrub and A. M., "Comparison Of Data Encryption Algorithms," *International Journal of Computer Science and Communication*, vol. 2, no. 1, pp. 125-127, June, 2011.
- [22] R. P. (Arya), U. Mishra and A. Bansal, "A Survey on Recent Cryptographic Hash Function Designs," *International Journal Of Emerging Trends and Technology in Computer Science (IJETTCS)*, vol. 2, no. 1, pp. 117-123, February, 2013.
- [23] W. Stallings, *Cryptography And Network Security principles and practice*, United States Of America: Prentice Hall, 2011.
- [24] A. Rawat and D. Agrawal, "An Enhanced Message Digest Hash Algorithm for Information Security," *International Journal of Recent Research in Electrical and Electronics Engineering (IJRREEE)*, vol. 2, no. 1, pp. 64-52, March, 2015.
- [25] A. K. Kasgar and M. K. Dhariwal, "A Review Paper of Message Digest 5 (MD5)," *International Journal of Modern Engineering & Management Research*, vol. 1, no. 4, pp. 29-35, December 2013.

## APPENDIX

CODE

TEST MD5.JAVA

```
import java.io.*;
```

```
import java.util.*;
```

```
import java.security.MessageDigest;
```

```
import java.security.NoSuchAlgorithmException;
```

```
import java.util.Formatter;
```

```
import java.util.Scanner;
```

```
public class TestMD5 {
```

```
    public static void main(String[] args) {
```

```
        try {
```

```
            Scanner user_input = new Scanner( System.in );
```

```
            System.out.println("Press 1 if you want to enter text, Press 2 for  
entering a file path");
```

```
            String option = user_input.next();
```

```
            String s, text = ""; // user text
```

```
            if (option.equals("1")) {
```

```
                System.out.println("Enter text to hash");
```

```
                text = user_input.next();
```

```
            }
```



```

else if (option.equals("2")) {

    // file path

    System.out.println("Enter file path");

    BufferedReader in = new BufferedReader(new
    FileReader ( user_input.next ( ));

    while ((s = in.readLine()) != null) {

        text = text + s;

    }

    in.close();

}

else {

    System.out.println("Invalid input. Exit");

    System.exit(0);

}

long startTime = System.nanoTime();

System.out.println("Data length is "+text.length());

System.out.println("Encrypted key:\n "+getMD5HashVal(text));

long endTime = System.nanoTime();

System.out.println("Complutation    took    "+(endTime    -
startTime)/1000000000.0 + "secs ");

}

```

```

        catch(Exception e) {
            e.printStackTrace();
        }
    }
}

```

```

public static String getMD5HashVal(String strToBeEncrypted) {
    String encryptedString = null, str = "", encryptedString1 = null;
    String DES_ENCRYPTION_KEY = "12345677";
    StringBuilder sb = new StringBuilder();
    byte[] bytesToBeEncrypted;
    try {
        for (int i =0; i < strToBeEncrypted.length(); i+=8)
        {
            int end_limit = i +7;
            if (strToBeEncrypted.length() < end_limit)
            {
                end_limit = strToBeEncrypted.length();
            }
            bytesToBeEncrypted =
strToBeEncrypted.substring(i,end_limit).getBytes("UTF-8");
            MessageDigest md = MessageDigest.getInstance("MD5");

```

```

        byte[] theDigest = md.digest(bytesToBeEncrypted);

        Formatter formatter = new Formatter();

        for (byte b : theDigest) {

            formatter.format("%02x", b);

        }

        encryptedString = formatter.toString().toLowerCase();

        String          encryptedMD5          =
Cryptography.encrypt(encryptedString, DES_ENCRYPTION_KEY);

        sb.append(encryptedMD5);

    }

    bytesToBeEncrypted = sb.toString().getBytes("UTF-8");

    MessageDigest md1 = MessageDigest.getInstance("MD5");

    byte[] theDigest1 = md1.digest(bytesToBeEncrypted);

    Formatter formatter1 = new Formatter();

    for (byte b : theDigest1) {

        formatter1.format("%02x", b);

    }

    encryptedString1 = formatter1.toString().toLowerCase();

}

catch (Exception e) {

    e.printStackTrace();

```

```
        }  
        return encryptedString1;  
    }  
}
```

#### TEST PROPOSED.JAVA

```
import java.io.*;  
  
import java.util.*;  
  
import java.security.MessageDigest;  
  
import java.security.NoSuchAlgorithmException;  
  
import java.util.Formatter;  
  
import java.util.Scanner;  
  
import java.util.logging.Level;  
  
import java.util.logging.Logger;  
  
public class Proposed {  
  
    public static void main(String[] args) {  
  
        try {  
  
            Scanner user_input = new Scanner( System.in );  
  
            System.out.println("Press 1 if you want to enter text, Press 2 for  
entering a file path");
```

```

String option = user_input.nextLine();

String s, text = ""; // user text

if (option.equals("1")) {

    System.out.println("Enter text to hash");

    text = user_input.nextLine();

}

else if (option.equals("2"))

{ // file path

    System.out.println("Enter file path");

    BufferedReader in = new BufferedReader(new
FileReader(user_input.nextLine()));

    while ((s = in.readLine()) != null) {

        text = text + s;

    }

    in.close();

}

else {

    System.out.println("Invalid input. Exit");

    System.exit(0);

}

long startTime = System.nanoTime();

```

```

        System.out.println("Data length is "+text.length());

        System.out.println("Encrypted key:\n "+getMD5HashVal(text));

        long endTime = System.nanoTime();

        System.out.println("Computation took "+(endTime -
startTime)/1000000000.0 + " seconds");

    }

    catch(Exception e) {

        e.printStackTrace();

    }

}

public static String getMD5HashVal(String strToBeEncrypted) {

    String encryptedString = null;

    String DES_ENCRYPTION_KEY = "12345677";

    MessageDigest md = null;

    try {

        md = MessageDigest.getInstance("MD5");

    }

    catch (NoSuchAlgorithmException ex) {

        //Logger.getLogger(Md5des.class.getName()).log(Level.SEVERE,
null, ex);

```

```

        ex.printStackTrace();
    }

    StringBuilder sb = new StringBuilder();

    byte[] bytesToBeEncrypted;

    try {

        for (int i=0; i < strToBeEncrypted.length(); i++) {

            bytesToBeEncrypted =
Character.toString(strToBeEncrypted.charAt(i)).getBytes("UTF-8");

            //This contains 128 bits

            byte[] bits_128 = md.digest(bytesToBeEncrypted);

            Formatter str64BitsFormatter1 = new Formatter();

            // 64 bits in first string

            for (int k=0; k < 8; k++) {

                str64BitsFormatter1.format("%02x", bits_128[k]);

            }

            String desString1 =
str64BitsFormatter1.toString().toLowerCase();

            //Applying encryption on first half

            String encryptedMD51 = Cryptography.encrypt(desString1,
DES_ENCRYPTION_KEY);

```

```

        Formatter str64BitsFormatter2 = new Formatter();

        // 64 bits in second string

        for (int j=8; j < 15; j++) {

            str64BitsFormatter2.format("%02x", bits_128[j]);

        }

        String desString2 =
str64BitsFormatter2.toString().toLowerCase();

        //Applying encryption on second half

        String encryptedMD52 = Cryptography.encrypt(desString2,
DES_ENCRYPTION_KEY);

        //Combining the two DES Strings

        String combined = encryptedMD51 + encryptedMD52;

        //Feeding that input to hashloop again

        md.update(combined.getBytes());

    }

    byte[] finalDigest = md.digest();

    Formatter finalFormatter = new Formatter();

    for (byte b : finalDigest) {

        finalFormatter.format("%02x",b);

    }

```



```
        encryptedString = finalFormatter.toString().toLowerCase();
    }
    catch (Exception e) {
        e.printStackTrace();
    }
    return encryptedString;
}
}
```