

Performance Evaluation of Message Encryption Scheme Using Cheating Text

Ch.Rupa,
Student, (Ph.D),
rupamtech@gmail.com

P.S.Avadhani,
Professor,
psavadhani@yahoo.co.in

Phone No: +91 {9848690640, 9885187815}
Department of Computer Science and Systems Engineering
Andhra University
Visakhapatnam, Andhra Pradesh, India

Abstract- In this paper we evaluate the performance of our proposed Message Encryption scheme using cheating text [6] with respect to response time and user load. In this method the authentication of the message is also possible because of the hashing by Modified message digest Algorithm [6]. This scheme can be applied for authentication like security in data bases. The response time of the algorithm is implemented in java is calculated with user load and the different sizes of the data file. The results are also compared with the performance evaluation done by Priya Dhawan [8], Aamer Nadeem [7] and found to be efficient.

I. INTRODUCTION

The popular method of authentication is by using a hash algorithm. MD5 is a hash algorithm to prepare a message digest for a given plaintext. However, this suffers from Wang's collision attack [2]. Modified Message Digest algorithm is modified to sustain the Wang's collision attack. The idea is to use 64-bit chaining variables instead of 32-bit. The performance of our MMD encryption scheme is evaluated and compared with the Priya Dhawan [8], Aamer Nadeem [7]. The results are shown in section 3 and section 4.

Keywords: Junit, MMD, Wang's Collision.

II. AUTHENTICATION SYSTEM EVALUATION

1) Performance Evolution Methodology

This section describes the techniques and simulation choices made to evaluate the performance of the compared algorithms. In addition to that, this section will discuss the methodology related parameters like: system parameters, experiment factors, and experiment initial settings. It includes an overview of Junit test which is used for finding the response time of the algorithms. The experiments are conducted using 64-bit processor with 1GB of RAM [6]. The simulation program is compiled using the default settings in Java, windows applications. The experiments are performed number of times to assure that the results are consistent and are valid to compare the different algorithms.

In order to evaluate the performance of the compared algorithms, the parameters that the algorithms must be tested for, must be determined since the security features of each algorithm and their strength against cryptographic attacks is already known [2]. The chosen factor here to determine the performance is the algorithm's speed to find the hash value with the proposed algorithm of the data blocks of various sizes. The response time of the algorithm is implemented in java is calculated with user load and the different sizes of the data file. By considering different sizes of data blocks (4KB,135KB) the algorithms were evaluated in terms of the time required to hash functions using modified message digest algorithm and All the implementations were exact to make sure that the results will be relatively fair and accurate.

III. TESTING FRAME WORK

J-Unit is a unit-testing framework that can be used to test the functionality of java classes [11]. Writing a test is a method that exercises the code to be tested and defining the expected result. The framework provides the context for running the test automatically and as part of a collection of other tests. This investment in testing will continue to pay us back in time and quality.

a. Need of JUnit

1. Junit is useful to write tests that exercise our code and incrementally add tests as the software grows.
2. Testing become tough if we have to manually compare the expected and actual result of tests, and it slows us down. JUnit tests can be run automatically and they check their own results. When we run tests, we get simple and immediate visual feedback as to whether the tests passed or failed. There's no need to manually comb through a report of test results.

b. Features of JUnit

1. For testing java classes written by us we need to write test classes using features of JUnit.
2. **TestCase** is a command Object. Test Classes must extend this **TestCase** class. A **TestCase** can define any number

of public `testXXX()` methods. When we want to check the expected and actual test results, we can invoke a variation of the `assert()` method.

3. TestCase subclasses that contain multiple `testXXX()` methods can use the `setUp()` and `tearDown()` methods to initialize and release any common objects under test, referred to as the test fixture. Each test runs in the context of its own fixture, calling `setUp()` before and `tearDown()` after each test method to ensure there can be no side effects among test runs.

4. TestCase instances can be composed into TestSuite hierarchies that automatically invoke all the `testXXX()` methods defined in each TestCase instance. A TestSuite is a composite of other tests, either TestCase instances or other TestSuite instances. The composite behavior exhibited by the TestSuite allows you to assemble test suites of test suites of tests, to an arbitrary depth, and run all the tests automatically and uniformly to yield a single pass or fail status.

IV. RELATED WORK

In this section discusses the results obtained from Priya Dhawan [8], Aamer Nadeem [7]. By considering the response time, complexity and security will improve with the Advanced Authentication Method. The following graph is taken into consideration by considering different User load. Dhawaan [8], Nadeem [7] has done experiments for finding the performance of MD5. The method computes the hash of data stored in a file. It performed the tests with a data size of 4 KB, 135 KB to see how the size of data impacts performance. The algorithms were implemented in a Java, using their standard specifications, and were tested to compare their performance.

Fig 1 and 2 gives the Existing experiments report for performance of 4 KB and 135KB data files with respect of response time.

Fig 3 and 4 gives the Modified Message digest Algorithm reports for performance of 4 KB and 135KB data files with respect of response time.

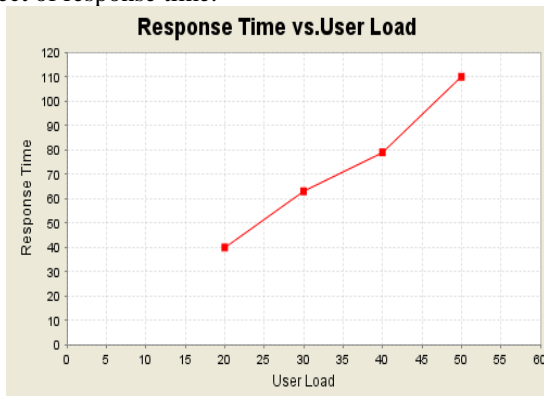


Fig 1. Graph for 4 KB data File

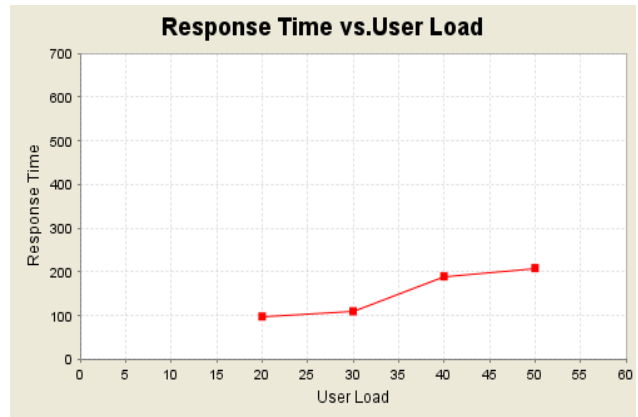


Fig 2. Graph for 135KB data File

V. PERFORMANCE METHODOLOGY ADOPTED FOR THE PROPOSED SCHEME

In this section a step by step procedure is given to illustrate the methodology adopted for the performance evaluation of the message encryption scheme proposed in [3,6].

Step 1: Developed unit test cases for the algorithm using JUnit Testing tool.

Step 2: Developed wrapper test cases using JUnitPerf testing tool for the test cases that were developed using JUnit as mentioned in step2.

Step 3: Using Load method of JUnitPerf, set the maximum number of concurrent users to execute Modified Message Digest [6] functionality simultaneously.

Step 4: Change in the response time can be observed with the change of user load (max number of concurrent users) with respect of executing the above steps.

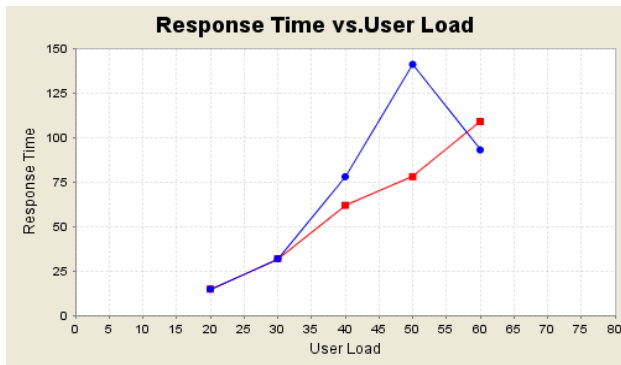
Modified Message Digest algorithm [3,6] is getting the results in 'milli Seconds'. Table 1 and Table 2 contains the speed benchmarks for the 4KB data file using proposed algorithm and Existing Algorithm. Table 3 and Table 4 contains the speed benchmarks for the 135KB data file using proposed algorithm [3,6] and Existing Algorithm [2,5]. Existing Algorithm [2] is having inconsistency as shown in Fig 3 along with the collision attacks [2]. Proposed algorithm [3,6] overcome the limitations what are existed.

Load	Time Taken(ms)
20	0.015
30	0.032
40	0.062
50	0.078
60	0.109

Table 1 Response time for 4 KB data file with respect of user load (in milli seconds)

Load	Time Taken(ms)
10	0.016
20	0.015
30	0.032
40	0.078
50	0.141
60	0.093
70	0.203

Table 2 Response time for 4 KB data file with respect of user load using Existing algorithm.



.....Existing Algorithm(MD5)
Proposed Algorithm(MMD5)

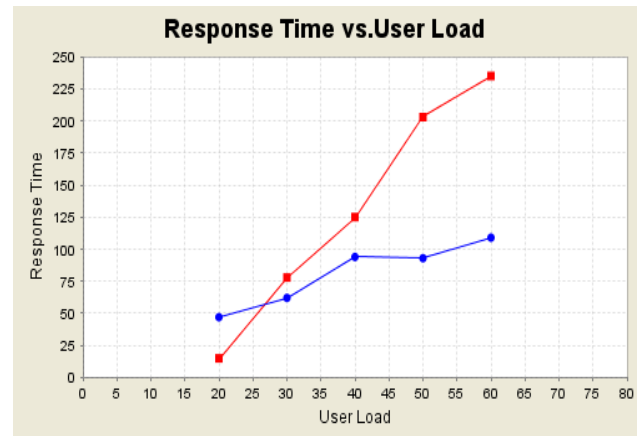
Fig 3 Comparison results for 4 KB files with the existing algorithm.

Load	Time Taken(ms)
20	0.015
30	0.078
40	0.125
50	0.203
60	0.235

Table 3 Response time for 135 KB data file with respect of user load (in milli seconds)

Load	Time Taken(ms)
10	0.016
20	0.015
30	0.032
40	0.078
50	0.141
60	0.093
70	0.203

Table 4 Response time for 135 KB data file with respect of user load using Existing algo



...Existing Algorithm(MD5)
Proposed Algorithm(MMD5)

Fig 4. Comparison results for 4 KB files with the existing algorithm.

VI. CONCLUSION

The presented testing results showed that MMD has a better performance . As these tests demonstrate, authentication

schemes and hashing algorithms carry varying amounts of overhead, and therefore have vastly different performance characteristics. The size of data being passed to hashing algorithms, as well to cryptography techniques, is also significant.

VII. ACKNOWLEDEMENT

I thank full to Prof P.S. Avadhani and Prof Nagalakshmi for his esteemed guidance and encouragement during all the time of this work.

REFERENCES

- [1] H. J. Highland, “**Data encryption: a non-mathematical approach-Part 5,**” Journal of computer and Security, pp.93-97, 1995.
- [2] Xiaoyun Wang, Dengguo Feng, Xuejia Lai and Hongbo Yu, “**Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD**”, Science and Communication, <http://eprint.iacr.org/2004/199.pdf>, 2004.
- [3] Ch. Rupa and P. S. Avadhani, “**An Improved Method to Reduce the Occurrence of Collision Attack on Hash Function**”, Int. J. computing mathematical applications, vol2, No1-2, pp.121-131, 2008.
- [4] H. Dobbertin, “**Cryptanalysis of MD5 Compress**”, proc. of Eurocrypt '96, 1996.
- [5] R.L. Rivest “**RFC 1321: The MD5 Message-Digest Algorithm**”, M.I.T. Laboratory for Computer Science and RSA Data Security, Inc., 1992
- [6] Ch. Rupa and P. S. Avadhani, “**Message Encryption Scheme Using Cheating Text**”, ITNG ISBN: 978-0-7695-3596-8/09(indexed by IEEE,dblp), pp. 470-475, 2009.
- [7] Aamer Nadeem et al, “**A Performance Comparison of Data Encryption Algorithms**”, IEEE information and communication, pp .84-89, 2005.
- [8] Priya Dhawan., “**Performance Comparison: Security Design Choices**”, Microsoft Developer NetworkOctober2002.<http://msdn2.microsoft.com/en-us/library/ms978415.aspx>
- [9] R Weis and S Lucks, “**Cryptographic Hash Functions-Recent Results on Cryptanalysis and their Implications on System Security**”, 5th System Administration and Network Engineering Conference, pp 15-19, 2006.
- [10] M. Bellare, R. Canetti and H. Krawczyk, “**Keying hash functions for message Authentication**”, Journal of Advances in Cryptology — Crypto '96, pp.1-15, 1996.
- [11] P. Louridas, “**JUnit: unit testing and coiling in tandem**”, Software, IEEE, Vol 22, pp.,12- 15, 2005.
- [12] Ch.Rupa, Prof P.S. Avadhani and Ch. Devi Chamundeswari, “**An Encrypto Stego Technique in wireless communication**”, IEEE – sipicom, pp. 28-30, 2006.