A Novel Methodology to Protect from Attacks using Multiple Hashing Algorithms

Boreddy Nikhil Reddy
Bachelor of Engineering
Manipal Institute of Technology
Manipal,Karnataka, India

Nimit Acharya
Bachelor of Engineering
Manipal Institute of Technology
Manipal,Karnataka, India

Kishore Bhamidipati
Asst. Professor – Senior Scale
Manipal Institute of Technology
Manipal, Karnataka, India

ABSTRACT
Many hashing techniques have been developed to secure data and store information like passwords, in the form of hash codes, which appear as a sequence of random characters. The hashing algorithms used have the pitfall that they can be reverse engineered using large tables that contain hash codes for frequently used passwords. In this paper a methodology is presented to protect against such attacks by using multiple hashing algorithms together.

General Terms
Security, Hashing

Keywords
Security, Hashing, MD5, SHA1 and Secure Passwords.

1. INTRODUCTION
In this age of Information Technology protecting the identity of an individual is of immense importance. The corporations spend a lot of resources to enhance the level of security for a client in order to protect his credentials. As the techniques to secure data become more advanced, even the techniques to breach security are advancing. So there is a need to constantly improvise the methodologies used to protect data.

Depending on the requirements as well as the usage of data, security is implemented through various methods. Encryption and hashing are two different techniques used to secure data or credentials in data warehouses. Encryption is used to secure messages during message transfer. Both the sender and receiver share a pass-phrase (key). Any encryption methodology changes the plain text into cipher which is totally random jumbled characters which make no sense. The plain text can only be got back after providing the privately shared pass-phrase to the appropriate algorithm used. So basically the sender encrypts the data and sends the cipher to the receiver who has to provide the pass-phrase in order to decrypt the cipher and view the plain text. On the other hand, hashing is a one-way cipher such that it is not possible to decrypt the cipher using any known technique. It is a message digest, meaning it generates a unique fixed length cipher for a particular input and which is not possible to reverse engineering provided you have only the cipher. Typically it is used to store passwords in the databases, such that only the hashed passwords are stored. When the user provides the plain text password it is hashed using the same algorithm and then compared with the one in the database and the user is authenticated.

In this paper an advanced hashing technique is presented to increase the level of security and to counter brute force attacks. There are few hashing algorithms which are commonly used such as MD5, SHA1, SHA2 etc. The main advantage of hashing is that even if the attacker manages to gain access to the database it is not possible to reverse engineer the hashes and to get back the plain text passwords.

But now it has become easier to crack the traditional hashing algorithms using rainbow tables and this is explained in detail in the next section so an advanced hashing technique is proposed which carries out hashing 2 times and also takes into account the unique user names to protect the integrity of the system.

In the following section, the traditional MD5 algorithm as well as SHA1 algorithms and how they work using examples are discussed. In section 3.0 a new methodology which makes use of both MD5 and SHA1 algorithms is described. After that in section 4.0 the results of the implementation and comparison with the results obtained using the normal systems in use are discussed. In 5.0 statistical data is provided thereby proving that implementation is feasible as well as advantages. This is followed by 6.0 where references are cited.

2. EXISTING SYSTEMS

2.1 MD5 Hashing
It stands for Message-Digest 5 Algorithm. It is a cryptographic hash which produces a 128-bit hash value. We interpret that value as a hexadecimal number which is 32 digits long.

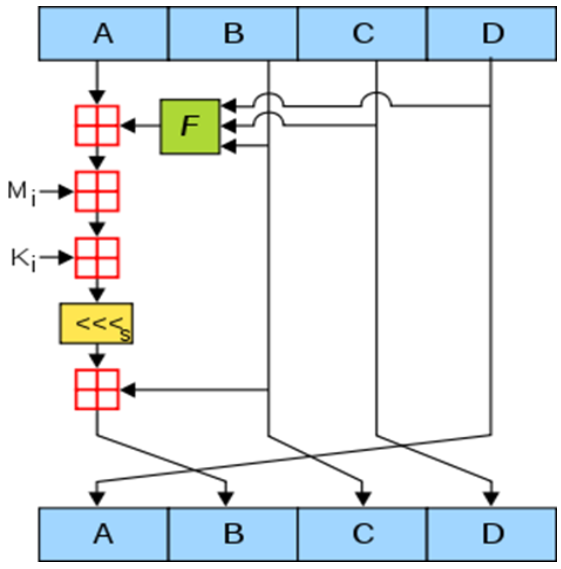

Fig 1: The above figure shows one operation of MD5[1]

MD5 processes a variable length input and generates fixed length 128-bit output. The input message is divided into chunks of 512-bit blocks. If the message length is not

divisible by 512 then it is padded to make it divisible by 512. Then if processes the equal chunks as shown above. And it basically uses a complicated bit manipulation using XOR gates to generate fixed length output.

2.2 SHA1

It stands for Secure Hash Algorithm. It was initially designed by United State National Security Agency. It produces an output of 160-bits which we interpret as 40 hexadecimal digits.

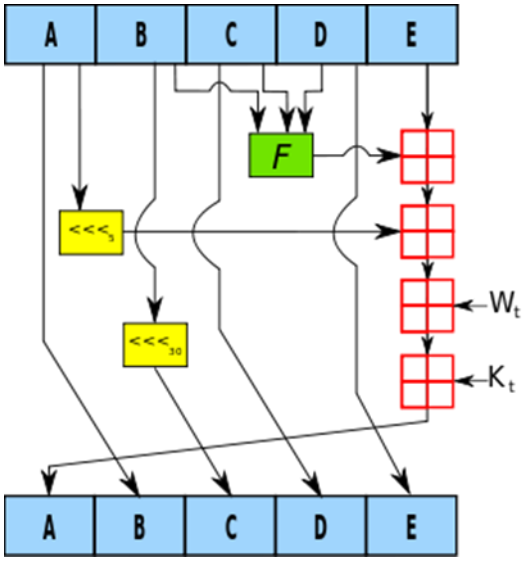

Fig 2: The above figure shows one iteration of SHA1[1] implementation

Even this algorithm accepts a variable length input and produces a fixed length i.e., 160 bit output. It processes a block with 512-bits for each iteration. It is more complicated computationally when compared to MD5 and the bit manipulation is achieved by the following operations add, and, or, xor, rotate and mod.

2.3 Problems With the Existing System

Most of the existing systems hash passwords using either of the explained hashing algorithms. But since the hashing algorithms like MD5 have been around since a long time, this has allowed the attackers to construct rainbow-table which are actually pre-computed tables for reversing cryptographic hashes. After getting the hash from the database the attackers simply look up the hash in the pre-computed table and obtain a match and hence the clear text password is revealed. Since it is impossible to construct rainbow-tables for all possible plain-text passwords, mostly the rainbow-tables are constructed for commonly used passwords or strings up to a certain length. An example for an existing rainbow-table is given below

Table ID	Charset	Plaintext Length	Key Space	Success Rate	Table Size
md5_ascii-32-95#1-7	ascii-32-95	1 to 7	70,576,641,626,495	99.9 %	64 GB
md5_ascii-32-95#1-8	ascii-32-95	1 to 8	6,704,780,954,517,120	96.8 %	576 GB
md5_mixalpha-numeric#1-8	mixalpha-numeric	1 to 8	221,919,451,578,090	99.9 %	160 GB
md5_mixalpha-numeric#1-9	mixalpha-numeric	1 to 9	13,759,005,997,841,642	96.8 %	864 GB
md5_loweralpha-numeric#1-9	loweralpha-numeric	1 to 9	104,461,669,716,084	99.9 %	80 GB
md5_loweralpha-numeric#1-10	loweralpha-numeric	1 to 10	3,760,620,109,779,060	96.8 %	396 GB

Fig 3: Example of MD5 Rainbow Tables

Judging by the size and success rate of MD5 we observed that present implementation of password only hashing is easy to breach. Similarly even there are methods to crack SHA1 which have been theoretically proved. More advanced techniques like SHA2 and SHA3 are being used for few applications and are more secure. But using the method of rainbow tables it is easy to crack the password generated by using such hashing algorithms primarily because domain of passwords is restricted in terms of length and no hashing algorithm is resistant to attacks via rainbow tables.

To counter these drawbacks in this paper we propose a methodology to increase the security and protect the database against attacks using rainbow-tables.

3. PROPOSED METHODOLOGY

The proposed model aims at overcoming the drawbacks of a single level hashing and making it practically impossible to crack the passwords. The algorithm works as follows:-

While user registration

• The user enters the user name and password along with other details

• The user name is checked for its uniqueness as there should not be any collisions in the database

• The password is hashed using MD5 and stored as hexadecimal digits in temporary memory

• The MD5 hash of the password is appended to the unique user name

• This entirely new string is given as input for SHA1 algorithm and a 20-byte output is obtained

• This 20-byte output is stored as 40-hexadecimal digits in the final database for future authentication of the user

While Log-In process

• The user enters user name and password

• The user name is first validated

• The hashed value corresponding to that user name is fetched from the database which will be subsequently used for comparison

• The password is hashed using MD5 Algorithm and 32-hexadecimal digit string is obtained

• This obtained string is appended to the validated user name and new string is formed

• This new string thus obtained is given as input for the SHA1 algorithm and the 40-hexadecimal digit output is obtained

• This output string is matched against the one fetched from the database and if they are equal the user's identity is established and he is given access according to his access rights

• If the authentication fails a suitable error message is displayed

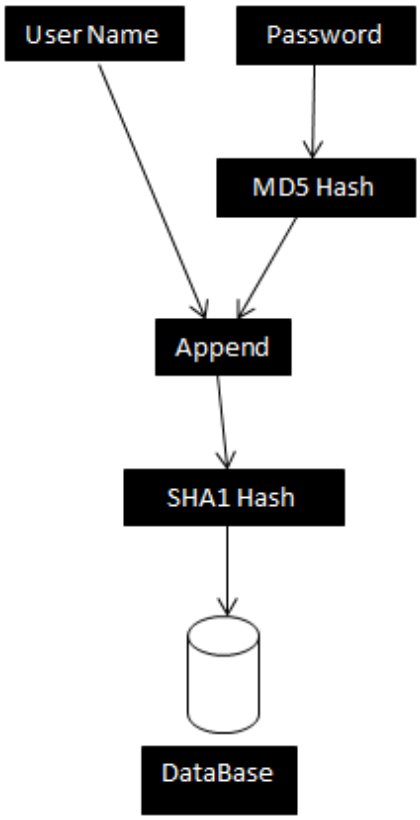

Fig 4: Operation of our proposed methodology

4. RESULTS

The implementation of the proposed model has been done in Java 7. In implementation a database of 2999 entries of user names and plain text passwords in which no 2 of them are same was taken. Then MD5 and SHA1 algorithms were applied separately only on passwords and stored the values in a different table which contained only these hashes and corresponding user name.

The results stated below are obtained from our implementation on a system with specification as follows:

• Windows 7 Enterprise 32-bit

• Intel Core 2 Duo @ 2.00 GHz 2.00 GHz

• IDE used : NetBeans 7.1.1

• Database Server : MySQL 5.5.24

The mean time taken for hashing 2999 plain text passwords from the data base using MD5 algorithm and storing the hashes along with corresponding user name in a table is 184555ms.

To perform the same using SHA1 algorithm it took 225907ms.

To hash the passwords using the proposed algorithm it took 308912ms on an average. This value was averaged over several iterations and maintaining the same system conditions. The snapshot given below shows the output from one such iteration which took 347879ms to hash and store the passwords.

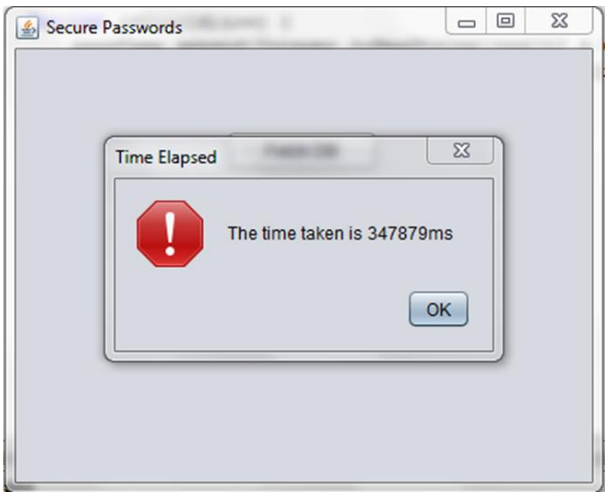

Fig 5: Snapshot to show the time taken in hashing

Example 1: Consider one of the entries in the database with user name as stark and password as start

Our approach

• Start is given as input to the MD5 algorithm and the intermediate hash string is ea2b2676c28c0db26d39331a336c6b92

• This intermediate hash string is append to the user name which is stark resulting in a string starkea2b2676c28c0db26d39331a336c6b92

• The above obtained string is given as input to SHA1 algorithm and the final hash is obtained which is 5f1605378bb61acc540b116506ac65abeb9598d8 in this case and this value is stored in the database

The following figures show the user name and password for first 10 records and the corresponding final hash obtained after applying out algorithm. It is impossible to obtain the plain text password from the finally hashed string using any existing rainbow tables.

userName	password
1952	sammy
stark	start
stack	stace
steve	stevy
steen	steet
pmc	1234
1235	1213
1211	maris
marie	marty
marth	macky

Fig 6: Sample Input

final
a181055599f84fafca79f11172de720a75e28916
5f1605378bb61acc540b116506ac65abeb9598d8
971e7b2b21f613cb4a0f472bd933b8c9078bc726
819b4690048027a43fe0a029b433036f93ff715e
75c2d02152632a2eee6454a4659a1c0e49202eb5
7c1b6a0f6211caf725af9ff011c5e26042342e3a
07c812999198d7a65543dd2335cb43cb568e052b
ee0efa55e667dd9ba60d70febcb05855c6010171
4e250d5893aa71487f921f60bed588724fa8158e
7836dbfc8addaba1586f9bf02535cb9d99a11a9a

Fig 7: Output obtained for the above given inputs

5. COMPARISON

Table 1. Time taken to compute hashes using different algorithms

Algorithm	Time Taken for 2999 entries(in ms)
MD5	184555
SHA1	225907
Proposed Algorithm (MD5+SHA1)	347879

Table 2. Number of possible hashes algorithm wise

Algorithm	Possible number of hashes
MD5	2^{32}
SHA1	2^{40}
Proposed Algorithm* (MD5+SHA1)	$2^{32}*2^{40}=2^{72}$

*assuming user name set covers a large yet unique alpha numeric set of characters.

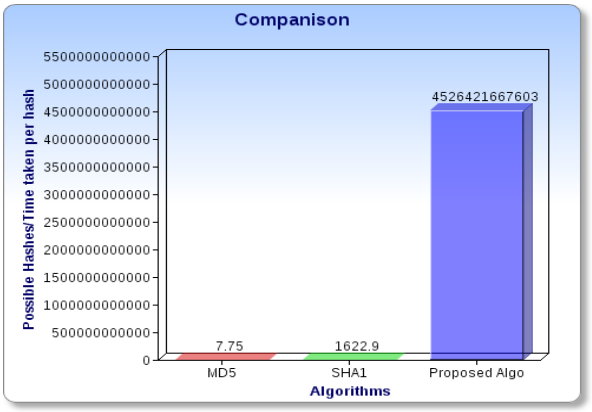

Fig 8: Possible hash space graph

The graph clearly shows the difference of the size of possible hashes using different algorithms. The difference indicates the superiority of the proposed algorithm with rest to the already existing ones.

6. CONCLUSION

By the hashing of the large database of 2999 records we observe that our algorithm does impose an overhead over a simple MD5 or SHA1 hashing algorithm but this overhead is extremely small for each record. This small overhead is needed to enhance the security and prevent attacks using rainbow tables. Considering the high computation speed of any modern computer the extra time taken for hashing and storing one record is almost negligible. In our algorithm basically a 40 character plus string is being hashed using SHA1 to product the final hash from this final hash it is impossible to obtain the intermediate hash because such a large rainbow table having records up to length 40 is not available and it is not feasible to construct such a big table even including the random arrangement of hexadecimal digits. The collisions that may occur in MD5 algorithm are taken care of by using a unique user name and adding it to the intermediate hash there by resulting in a unique final hash no matter what the input is.

7. REFERENCES

[1] Wikipedia
[2] VH Shear, WO Sibert, DM Van Wie - US Patent 6,292,569, 2001 - Google Patents. Systems and methods using cryptography to protect secure computing environments
[3] M Bagnulo, J Arkko, Support for Multiple Hash Algorithms in Cryptographically Generated Addresses (CGAs) - 2007
[4] Hoffman, P. and B. Schneier, "Attacks on Cryptographic Hashes in Internet Protocols", RFC 4270, November 2005.
[5] Bellovin, S. and E. Rescorla, "Deploying a New Hash Algorithm", NDSS '06, February 2006.
[6] Aura, T., "Cryptographically Generated Addresses (CGA)", RFC 3972, March 2005.
[7] Bagnulo, M. and J. Arkko, "Cryptographically Generate Addresses (CGA) Extension Field Format", RFC 4581, October 2006.
[8] Arkko, J., Kempf, J., Zill, B., and P. Nikander, "Secure Neighbor Discovery (SEND)", RFC 3971, March 2005.