ISSN: 0974-2115
www.jchps.com Journal of Chemical and Pharmaceutical Sciences

# Effective Password Hashing for Extended Security

**Sudhakar, Shawn Anston, P.Vaisagan**
Department of Information Technology, Jeppiaar Engineering College, Chennai, Tamilnadu - 600 119
***Corresponding author: E-Mail: vaisagan@gmail.com**

**ABSTRACT**

A password catalog that has been hacked can be decrypted using rainbow tables or brute force methods based on the type of encryption. This paper puts forth a method to prevent the use of rainbow tables to decrypt hashed data, making databases more secure.

**KEY WORDS**: Database, security, password, hash, cryptanalysis, rainbow tables, implicit security, polynomial.

## 1. INTRODUCTION

With the introduction of the computer, the dependency on computer systems for various functionalities has increased. The early computer that ran simple applications did not need much security as only trained personnel would be able to use it. The transition of the computer as a business machine to homes increased the number of users. This boom in computer users has increased the need for security.

In recent times the cases of cyber theft have increased. Many popular websites used by thousands of people have been hacked and their passwords leaked. The usual method of registration involves selecting a username and assigning a password to it. The password and username are stored in the database in an encoded format. If the database is hacked, the hacker retrieves the encoded data into his system and he only needs to try out the different decoding methods available before finding out the plaintext. Thus the job of a hacker is easier after he retrieves the encoded data from the database. This paper puts forth a method to provide an additional layer of security even after the encrypted data is retrieved from a database.

A hacked table contains hash values encrypted using MD-5 or other similar algorithms. The Hash value is not easily decrypted using traditional methods. But with the advent of rainbow tables it takes lesser time to find the decrypted value of the hash function.

During authentication of the user details the username and password are enter as it is done traditionally. The system uses the username and hashes it accordingly and finds the match in the database.

In our approach the password is divided into multiple parts and stored in multiple servers. Before storing into the server an encryption method is applied to the data. This may be extra work during normal authentication but provides security if the database is breached. Even if the encoding method is known, the hacker will not be able to find out the password combinations easily. In a website having large number of users there is a very large possible combination of usernames. The hacker has to find the actual usernames and passwords in this list of possible combinations. The data is stored in separate databases randomly so the data cannot be reconstructed without obtaining all the partitions.

**Proposed data partitioning system**

**Algorithm 1a.** (k, k) Data Partitioning Scheme

Input← data d

1. Choose randomly and uniformly from a finite field Zp, k – 1 numbers r1, r2…, rk-1.
2. Compute $r_k = d.(r_1 . r_2 \ldots . r_{k-1})^{-1} \bmod p$.
3. Construct the kth degree polynomial:

$p(k) = (x-r_1)(x-r_2)\ldots\ldots(x-r_k) \bmod p$

$= x^k + a_{k-1}x^{k-1} + a_{k-1}x^{k-2} + \ldots.. + a_1 x + a_0 \bmod p.$

4. The roots $r_1, r_{2\ldots}r_k$ of the polynomial p (k) represent the data partitions.

Output→vector R = [$r_1, r_2, \ldots, r_k$]

As seen from the above algorithm, partition creation requires k multiplications and one inversion operation modulo prime p.

Hence the time complexity of O(K).

Similarly, data reconstruction requires k multiplications which are illustrated by the following algorithm.

**Algorithm 1b.** (k,k) Data Reconstruction

Input←vector R= [r1, r2. . . rk]

1. Data d = r1, r2… rkmod p.

Output→d

**Example:**

Let data $d = 10$, prime $p = 31$, and let $k = 3$. We need to partition the data into three parts for which we will need to use a cubic equation, $x^3 + a_2x^2 + a_2x - d \equiv 0 \bmod p$.

Assume, $(x - r_1)(x - r_2)(x - r_3) = 0 \bmod 31$. We randomly choose two roots from the field, $r_1 = 19$ and $r_2 = 22$. Therefore, $r_3 = d \,.\, (r_1 \,.\, r_2)^{-1} = 10\ (19 \,.22)^{-1} \bmod 31 = 0.23$ . The equation becomes $(x - r_1)(x - r_2)(x - r_3) = x^3 - 21x^2 + x - 10 \equiv 0 \bmod 31$, where the coefficients are $a_1 = 1$ and $a_2 = -21$ and the partitions are 0.23, 22 and 19.

**Encryption Methods:** The first passwords were stored into databases as plaintext but with the advent of hacking techniques encryption techniques came into vogue. The most common encryption techniques used today are Message Digest-5 (MD5), Secure Hash Algorithm-2 (SHA2), and NT LAN Manager (NTLM) etc.

The Secure Hash Algorithm (SHA) was developed by the National Institute of Standards and Technology (NIST) and published as a federal information processing standard (FIPS PUB 180) in 1993, a revised version was issued as FIPS PUB 180-1 in 1995 and is generally referred to as SHA-1. The algorithm takes as input a message with a maximum length of less than 2 64 bits and produces as output a 160-bit message digests. The input is processed in 512- bit blocks. The heart of the algorithm is a module that consists of four rounds of processing 20 steps each. Further improvements were made to remove alleged weaknesses in the algorithm. The new version was released under the name SHA-2.

SHA-2 includes significant changes from its predecessor, SHA-1. The SHA-2 family consists of six hash functions with digests (hash values) that are 224, 256, 384 or 512 bits: **SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, and SHA-512/256.** The algorithms were first published in 2001 in the draft FIPS PUB 180-2, at which time public review and comments were accepted. In August 2002, FIPS PUB 180-2 became the new Secure Hash Standard, replacing FIPS PUB 180-1, which was released in April 1995.

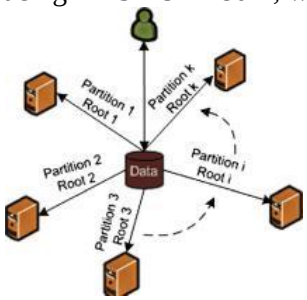


**Figure.1.Data partitioning system**

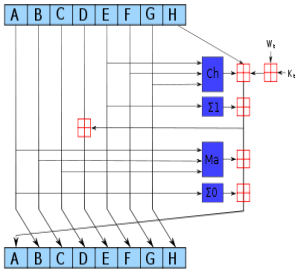


**Figure.2. SHA-2 processing of a single 512-bit block**

These cryptographic functions have become a bit insecure today with the rise of new decryption techniques like rainbow tables. MD-5 and SHA1 and their improvements suffer from collision vulnerabilities. These vulnerabilities can be mitigated with the use of salts.

**Decryption techniques available:** Various decryption techniques are developed along with the rise in encryption techniques. Some of the successful decryption methods used are discussed here.

Brute force attacks are the widespread form of decryption techniques used. They are generally ineffective if the user has a strong password. They can splinter a password if it is a very common and easy dictionary password.
The most basic precomputation technique is to compute and store the hashes of all the passwords in the keyspace ordered by hash, so that cryptanalysis is almost instantaneous. However, this requires storage equal to the keyspace size. Hellman (1980), showed how to decrease storage requirements at the expense of attack time in the general context of cryptanalyzing a cipher. This tradeoff works as follows. Given a fixed plaintext P, define a mapping fon the key space K as f(k) = R (E k (P)) where E is the encryption function andR is a reduction function which maps cipher texts to keys. By iterated applications off, we create chain" of keys. The crucial observation is that by storing only the first and last elements of a chain, we can determine if the key corresponding to a given cipher text belongs to that chain (and also find the key) in time O (t) here t is the length of the chain.

Creating a chain works as follows. Given a starting point k 0, we compute $K_1 = f(k_0)$, $k_2 = f(k_1)$, . . , $k_t = f(k_{t-1})$. The keys $k_0$ and $k_t$ are stored, whereas the rest are thrown away. When given a cipher text C to cryptanalyze, we re- cover the key by computing $k = R(C)$ and $k\,i = f_{i-i}(k)$ for i = 1, 2, . . . . Observe that if the key kbelongs to the chain, i.e, $k = k_i$ for some i, then $f_{t-i}(k) = k_t$. Thus, after at most t applications of f, we can determine whether or not the chain contains k. Further $i - 1$ applications of f suffice to compute $k_i - 1$ from k 0. Since $k_i - 1$ satisfies the property that $C = E\,k_i - 1\,(P)$, it is the key we are looking for. This does not work with certainty because different chains may merge. Therefore, we need to use multiple tables with a different reduction function for each table, with many chainsin each table. The storage requirement as well as the crypt-analysis time for this algorithm areO $(|K|^{2/3})$, whereKisthe key space.

A major progress was made by Oechslin (2003), by using"rainbow chains" instead of distinguished points. Rainbowchains use a different reduction function for each point in thechain. It was shown in Oechslin (2003), that rainbow chains achieve thesame coverage as distinguished points with the same storagerequirement but a significantly faster cryptanalysis time. Experimental results are remarkable. The online implementation of the rainbow attack

inverts MD5 hashes ofpasswords of length up to 8 over the character set[a-z0-9](a key space size of 2.8×1012). Its success probability is 0. 996with an amortized cryptanalysis time of under 10 minutesusing precomputed tables of size about 48 GB.

The SHA-2encryption technique cannot be easily decrypted with a brute force attack. A new method generated for decrypting MD-5 hashes is by using rainbow tables. Rainbow Tables are time-memory tradeoff technique that reduces the time taken to decrypt a password with the increase in memory. The rainbow table is a set of reverse hashed values using a series of reductions for a single hash value of a fixed amount of bit length. The values that are reduced are stored and used for future reference. This is an effective method as the time required is reduced with the increase in memory Hellman (1980).

The rainbow attack can be prevented by increasing the number of bits of the hashed value by appending randomized salt values to it. This makes it harder for the rainbow attack to succeed as the exact plaintext cannot be encrypted easily as an unsalted hash value.

**Proposed system's working:** The use of salts to increase the hash length is an efficient way to deter rainbow table attacks but cannot prevent it completely. The use of salting or key stretching techniques also enlarge the memory used to store the hashed value and the memory of the hashing program.

An effective way to prevent the use of rainbow attacks is to have a divided password hashing. The user given username and password are taken, in that the password is divided into two or more parts based on the hashmethod. Since they are separately hashed the rainbow attack cannot be done easily. A randomized salt is not necessary as the password separation technique itself works as a salt value that is encrypted. The encryption can be done using diverse techniques for each divided segment of the password for a stronger resistance to rainbow attacks.

In a rainbow attack the hashed value is decrypted using reduction functions for the whole hash value. In this method, when the hash values are computed separately using different algorithms it makes it more complex as the attack will not be able to guess which part of the hashed value uses what type of encryption algorithm. This method is more effective in preventing attacks as the speed of computing increases with time and can lead to more effective attacks even when salting is used.

**Implementation:** The following shows the authentication method used when a segmented hash is used for protecting an account.

**Segmentation:** The data given by the user can be segmented to two or more parts based on the proposed data portioning system. These segments are necessary to be of same or different lengths based on a dynamic algorithm that divides based on the length of the given input username and password.

**Methods and Materials used for Segmentation Algorithm:**

Let us consider that user has entered the username as "USERNAME" and password as "PASSWORD".

Username = USERNAME

Password = PASSWORD

The nearest square to the total number of characters is chosen. In the above example both counts the same as '8'. So the nearest square number '9' was chosen on square rooting that number gives '3'. Thus the given password is divided into 3 parts. Each part having 3,3, 2 letters respectively. Each letter gets partitioned that hides the original data and can be obtained only by having all parts.

**Results and discussion of partitioning data method:**

**Partitioning of data:** By using partitioning data method each letter in each part is provided with k root values, among them k-1 root values ($r_1$, $r_2$… $r_{k-1}$) are chosen randomly from a range Z and the kth value is calculated using the partitioning equation. Here let k=3 hence while finding 3 roots the following equation is used:

$r3= d.(r1 . r2)^{-1} \bmod p$

where r3, r2, r1 are root values

d is data value which is the ASCII value of a letter

p is the prime number which should be greater than d (p>d).

**Encryption:** The SHA-2 hashing method used here for each root of a segment before storing it into the database. When the hash function is completed the hash values are stored in select databases.

The total number of bits can be reduced by performing another hash function on it but it makes the hashed data ineffective. This is because the rainbow table will have a reduced set of plaintext data to traverse from if the length of the hash data is decreased. So it is generally not used.

This is similar to a normal password authentication but with an extra overhead in segmenting and comparing the hash values.

**Database selection:** In this the database is selected based on the division method explained above. Based on the square root value of the nearest square number the database is selected to store the segment data with the username. Then the second segment is stored in the next database and so on.

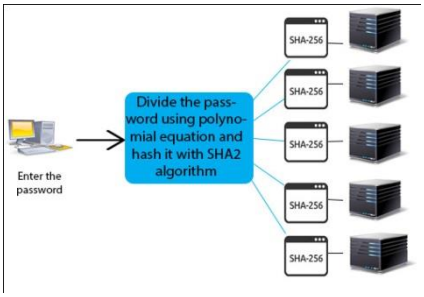


**Fig.3.Segmented Password Authentication**

## 2. CONCLUSION

The most common method of attacks are dictionary, brute-force and rainbow table attacks. These are the main type of attacks that a hashing system must be defiant to. Any hash value can be decrypted with enough time. With the increase in processor speeds every decade current attack methods can be highly successful. It is only logical to increase the confrontation to such attacks for gradual increase in processing speeds. The usage of memory is not a problem these days as it can be obtained easily but time cannot be obtained. Thus rainbow tables are used to reduce the total time taken to splinter a hash value with increase in memory use. The segmented hash method helps increment the time taken by rainbow tables to crack a hash value to increase with time without using extra memory space as in the case of a salted hash value.

## REFERENCES

Denning D, Cryptography and Data Security, Addison-Wesley, 1982.

Hellman M.E, A cryptanalytic time-memory trade-off, 1980.

Iyengar A, Scalability of dynamic storage allocation algorithms, Frontiers of Massively Parallel Computing, Proceedings Frontiers '96, Sixth Symposium on the, 1996, 223-232.

National Institute of Standards and Technology, Federal Information Processing Standards Publication 180-2, Secure hash standard, Technical report, National Institute of Standards and Technology (NIST), August 2002.

National Institute of Standards and Technology, Secure Hash Standard (SHA-1), Federal Information Processing Standards Publication #180-1, 1993.

Oechslin P, Making a faster cryptanalytic time-memory trade-off. In Proc. CRYPTO, 2729 of LNCS, 3, 2003, 617–630.

Parakh A and Kak S, Online data storage using implicit security, Information Sciences, Vol. 179(19), 2009, 3323-3331.

Rivest R, The MD5 Message-Digest Algorithm, RFC 1321, IT LCS & RSA Data Security, Inc, April 1992.