

AADHAR VERIFICATION USING SALT AND PEPPER

¹R. Nikhil Ram Reddy, ²Ashwin Sivaraman, ³P. Visalakshi^{1,2}B. Tech CSE, ³Assistant Professor, SRM University, Chennai, India¹nikdude80@gmail.com, ²ashwinsr20@gmail.com, ³visalakshi.p@ktr.srmuniv.ac.in

Abstract: Whenever a user enters his/her password for the first time, the database stores this password in hashed context. Storing of passwords as plaintext can pose a risk to security as malicious hackers can gain access to these passwords in the event of the database being compromised. Therefore, storing of passwords or other sensitive information in the database must be done only after proper hashing has been done. However, there is a degree of risk associated with the above methodology as the plaintext password can be retrieved by using rainbow tables (precomputed tables used for reversing cryptographic hash functions). To combat this, we hash the password combined with a randomly generated string of alphanumeric variables called SALT. SALT is prepended to the password along with PEPPER (a random letter from A-Z) and then the hash is calculated, which is further stored in the database. Therefore, when the user wants to login again, his details are verified in the following manner: The username and password entered by the user is prepended and appended with the salt stored in the database. Then we append pepper, followed by which the hash calculated and encrypted using AES and stored. Finally, decrypt the AES and compare with hashed password entered by user, if they both are equal then the user is authenticated.

Keywords: AES, login, authentication, encryption, decryption; database, password security, hashing, pepper.

1. Introduction

Hashing algorithms are one-way functions that turn any amount of data to a fixed length fingerprint i.e. the plaintext cannot be retrieved from the ciphertext. A small change in input results in considerable change in ciphertext. The following are the two main criteria to be met for a good hashing algorithm:

- The reverse of hash is computationally infeasible.
- The relation between input and hash is unpredictable.

Some examples of hashing functions are Message Digest and Simple Hashing Algorithm.

The following is an example of a hashing function: hash ("hello") can be 2caweqwd23awd34 and hash ("Hello") can be wd99q2e1q2easdasc. Although it is important to note that the size of a hashing function output should always be the same number of bits or letters.

A major threat posed towards database security in modern times are SQL injection attacks that use malicious code to manipulate the database into revealing information to unauthorised users. In the case of such breaches in security, the hashing of passwords grant an additional layer of security that renders the data obtained by the hacker useless. However, with the help of rainbow tables, that are essentially listings of all possible plaintext permutations of encrypted passwords specific to a given hash algorithm, the attacker may be able to obtain the passwords in database in plaintext format. Rainbow tables provide a greater degree of efficiency as compared to earlier methods such as brute force and dictionary attacks. An example of software that generate rainbow tables to be used in the cracking of passwords are RainbowCrack and ophcrack. In order to render such attacks useless, we have developed an improved version of password security. To remove these problems, we have come up with a better and improved solution to include salt and pepper in the plain text before hashing it. Salt could be a random n digit alpha numeric string that we add to the password at the front and back. We further add a pepper, which is a alphanumeric value to be appended to the string. For the sake of simplicity, we have chosen the set of strings to be the alphabets of English vocabulary. Finally, after adding both salt and pepper to the plaintext password, we hash it using a hashing algorithm, hashing algorithm chosen here has a low time complexity as well as space complexity as it has only one round of operation and less number of computational loops. After successful completion of the hashing process, we encrypt the hashed string for further safety using the most secure symmetric cipher algorithm (AES) and then we store the encrypted string in the database. At the time of validation of the authenticated user. We prompt the user to enter the user name and password. Then we add the salt and pepper to the

password. The value of the Salt is stored as a file in the database and can be added to the password and username easily. However, Pepper on the other hand is not stored anywhere in the database and gives a great degree of versatility to our security system. What we do is to take the chosen set of strings, that is the letter from A to Z and check the password for a match with each consecutive set of strings. In the case of a match, we move to the next step of the system. We do so by calculating the hash of this value one by one and check if it matches the one in the database. We repeat the above process until we either exhaust the alphabets or we reach match. If match we provide the user with access. While we use a rather obvious set of strings i.e. the letters in the English alphabet in order to match the pepper in the prototype version, in practice, any set of strings may be used.

In light of the recent splurge of attacks by hackers who have gained access to Aadhar databases that possess highly sensitive personal information of users such as biometrics and demographic data that can be used to malicious ends by the attacker. Our system aims to counter such attacks.

2. Algorithm

A. Overall Process

- The string is appended and prepended with a random six-digit alpha numeric SALT.
- Store the SALT.
- Append the resultant string with a randomly chosen letter from A to Z as PEPPER.
- Do not store PEPPER anywhere.
- Hash the resultant and store in database for verification.

B. Hashing Process

- Append padding bits such that length of message is 448 modulo 512. starting with zero and adding ones.
- Append block of 64 bits, which contains the length of original message to the resultant.
- Initialize a buffer called A.
- Perform one round by taking the resultant string and mixing it with Buffer A and XOR them.
- Store the resultant as A.
- Do the above steps for all bits in message, the final buffer A is the hashed string.

C. Encryption and Decryption Process

- After getting the hashed string which is of 512 bits.
- Start by encrypting the hashed string with the help of AES algorithm.

- After encryption of the hashed string store it into the database.
- Only at the time of verification of the authentic user, the encrypted value in the database is retrieved.
- After retrieval of the encrypted value pass it through the decrypting function of AES for decrypting the encrypted hashed value into the hashed value.
- Finally, compare the hashed value from the database with the hashed value generated from the credentials entered by the user.
- If they are same, the user is given access.

D. AES algorithm

- Advanced Encryption Standard published by National Institute of Standards and Technology (NIST) in the year 2001.
- AES is a symmetric block cipher.
- Cipher takes a plain text block size of 128 bits or 16 bytes.
- The key length can be 16, 24 or 32 bytes
- We are using AES-256, key size=32 bytes.
- Number of Rounds = 14 depends on the key size.
- Round 0: Initial Transformation
- First N-1 rounds consists of four distinct functions SubBytes, ShiftRows, MixColumns and Add Round Key.
- Final Round consists of three transformations.
- Each Transformation takes one or more 4x4 matrix.

E. Generation of Aadhar number

- To generate a unique Aadhar number we use the following famous algorithm called Luhn algorithm.
- It is a famous checksum algorithm used currently to verify credit cards and debit cards number.
- Aadhar number is a 12-digit number.
- First 11 digits are selected randomly by the computer.
- Then the last digit is the checksum digit calculated using Luhn algorithm.
- The algorithm states that we need to add all the digits from left to right one by one.
- If we encounter the sum to be more than 9 we need to subtract 9 from the sum.
- The final generated checksum is added to the end of the 11-digit aadhar number.
- Hence, we have a total of 12-digit unique aadhar number.

3. Literature Survey

Security of Aadhar applicants has been of a concern in the recent times. Applicants are worried whether their data is safe or not. To increase the security of the

information provided by the applicants this project has been done as a demo. Storing of applicant's data like name, age, address, etc in a database is quite risky as if the database gets compromised the whole data is compromised. Hence a separate way to store data in the database has been demonstrated here in this project.

A. Some Commonly Used Protection Methods

Normally most of the databases store the data as raw data and then access it when it is needed. This is really dangerous as any person with access to this database can view or modify the raw data. Some of the databases store the data after hashing it, this provides some kind of safety but is vulnerable to rainbow tables.

B. Proposed Method of Protection of Data

- We propose that instead of Storing data as raw data or even storing it as hashed string. We need to perform two special operations first.
- Append and prepend a random alpha numeric string to the raw data, this string is called as salt. Store this salt in the data base as we will need it later for retrieval of data.
- Add a randomly chosen letter from A-Z as pepper to this string. This string is not stored anywhere and hence even if the database gets compromised the raw data cannot be recovered.
- Finally hash the resultant string and encrypt it for storing it in the database. This kind of protection is infeasible to reverse.

C. Equations

Some commonly used equations are specified in the below section. These include equations regarding the hashing algorithm along with the salting and the addition of pepper to the message string. First, we modify the message by adding a random six letter alpha numeric to the front and back of it. Then we add a randomly chosen letter from A-Z to the end of it. Finally, we hash the resultant string and encrypt it for further security purposes.

- (1) $string = salt + message + salt + pepper$
- (2) $H = hash(string)$
- (3) $encrypt(H)$

D. Reason for Use of Salt And Pepper

- The usage of salt allows us to randomize the raw data more in spite of hashing it.

- Salt being a random set of alpha numeric string changes for every set of raw data, therefore it is tough for the attacker to find out which set of data uses which salt.
- The only vulnerability of using salt is that it has to be stored in the database for verification of data.
- This vulnerability of salt leads to SQL injection attacks and furthermore leads to compromise in security.
- Therefore, we include a special letter called pepper in the string.
- Pepper consists of a letter chosen randomly from A-Z.
- This randomly chosen letter is added to the end of the string and then hash is calculated.
- The specialty about pepper is that it does not get stored anywhere.
- Then how do we authenticate the user?
- We authenticate the user by simply choosing every possible letter from A-Z during authentication and adding to the provided string and then hashing it.
- We repeat the above process until we either get the match or we exhaust the letters from A-Z. In the latter case the user is denied of authentication and In the former case the user is provided with data.

4. Results

The demo program to illustrate the security enhancement to store raw data of Aadhar applicants has been implemented in python 2.7.14 build. The advantages of using Python include the presence of extensive support libraries, open source and community environment and the presence of third party modules to name a few.

A. Analysis of security

The proposed model has been analysed using various known attacks and their response is provided below.

1) Brute force attack

Brute force attacks are common to be the first to be used to crack any given hashing algorithm. To improve the security of the hash function we used the salt and pepper combination to make it totally infeasible to obtain the raw data without proper permission by anyone. Suppose the user selects a password "hello" this would then be used to create a string like "123jhqhellojopq21" this string is then changed again into " 123jhqhellojopq21p" by addition of random pepper. This final string is then hashed to provide a safe and secure value which cannot be cracked even with the help of rainbow tables as the value of the pepper is not stored anywhere in the database.

2) DoS Attack

Denial of service attacks are done to prevent other users from getting access to the services of the application. It is done by overwhelming the server by sending too many empty requests to a server repeatedly to ensure that the server is not able to answer any of the requests and shuts down as a result. This system monitors the user entries and if the user tries to attempt more than three entries of wrong password, the system suspects possible Denial-of-Service attack and blocks the user that is trying to login. This way we can prevent DoS attacks. Most of the operation involved in the application is bit oriented therefore if a malicious user tries to login using wrong details more than three times the system blocks the user and logs an entry about malicious attempt made by the respective IP address along with the time.

3) SQL Injection Attack

In the event of the hacker obtaining the sensitive data from the database, the hacker will only be able to see the encrypted text. Even if the user manages to decrypt the text there is still the hash included with salt and pepper in the text.

The value of the pepper is not present anywhere in the database and hence the user will not be able to obtain the plaintext from the database.

5. Conclusion

The paper deals with the security of the user data by means of securing the username and password of the user which would also eliminate the need to encrypt the raw data of the user entirely. This would ensure that the user has optimum security from malicious attacks. The customised hashing algorithm used in the project is inspired by the message digest algorithm. This enabled us to improve the performance of the previous algorithm and make them more efficient in future applications.

6. Future Scope

Improvements can be made in our project with regard to certain features. The addition of a more complex hashing and encryption algorithm along with betterments in time and space complexity can be added to enhance the functionalities of our existing project.

7. Acknowledgement

I would like to thank my project guide for her guidance and support through the duration of development of the

project. I would also like to thank all the teachers from my University who have lent their ideas and expertise that made this project possible.

References

- [1] <https://crackstation.net/hashing-security.htm>
- [2] <https://gooroo.io/GoorooTHINK/Article/13023/The-difference-between-encryption-hashing-and-salting/2085#.WeN9s2iCyUk>
- [3] <https://www.addedbytes.com/blog/why-you-should-always-salt-your-hashes>
- [4] <https://stackoverflow.com/questions/8305893/what-is-the-advantage-of-salting-a-password-hash>
- [5] An Improved Hashing Based Password Security Scheme Using Salting and Differential Masking- iee paper
- [6] [https://en.wikipedia.org/wiki/Pepper_\(cryptography\)](https://en.wikipedia.org/wiki/Pepper_(cryptography))
- [7] <https://security.stackexchange.com/questions/3272/password-hashing-add-salt-pepper-or-is-salt-enough>
- [8] <http://www.barkingiguana.com/2009/08/03/securing-passwords-with-salt-pepper-and-rainbows/>
- [9] T. Padmapriya and V. Saminadan, "Inter-cell Load Balancing Technique for Multi-class Traffic in MIMO - LTE - A Networks", International Conference on Advanced Computer Science and Information Technology, Singapore, vol.3, no.8, July 2015.
- [10] S.V.Manikanthan and V.Rama "Optimal Performance of Key Predistribution Protocol In Wireless Sensor Networks" International Innovative Research Journal of Engineering and Technology, ISSN NO: 2456-1983, Vol-2, Issue –Special –March 2017.

